Sammanfattning

Interaktiva WWW-tillämpningar blir mer och mer vanliga och får en allt viktigare del i många företags informationshantering. I dessa tillämpningar spelar databasaccesser en central roll. Tillvägagångssättet är ofta att HTML-dokument genereras dynamiskt utifrån information i en databas och med hjälp av framförallt HTML-formulär ges möjlighet att uppdatera databasen med ny information. VRML[1], som granskas i SISU Rapport 96:15 – *Affärsapplikationer i 3D på Internet,* möjliggör en helt ny typ av kraftfullare WWW-system. Men för att dessa nya interaktiva VRML-system skall få en riktigt stor spridning och hitta de stora applikations-områdena krävs även här möjligheter till databasaccesser. Det måste vara möjligt att dynamiskt generera VRML-världar samt att i vissa fall även uppdatera information i databaser från VRML-världen.

För att utreda VRMLs möjligheter närmare har SISU tagit fram en VRML-applikation som utnyttjar merparten av den avancerade funktionalitet som VRML 2.0 erbjuder idag. VRML-världen i applikationen som beskrivs i denna rapport genereras dynamiskt och användaren ges möjlighet att lagra den aktuella konfigurationen av VRML-världen till databasen. VRML-applikationen är fritt åtkomlig över Internet. Applikationen har en hög grad av interaktivitet och kan manipuleras direkt inuti världen. Förutom möjligheterna till interaktion inuti världen innehåller applikationen en från världen fristående Java-applet som kommunicerar med och kan styra världen.

Med applikationen och erfarenheterna från utvecklingsprocessen som grund förklaras i rapporten hur man går till väga för att lägga till liknande funktionalitet i sina egna VRML-applikationer. Varje väsentlig del av applikationen presenteras och förklaras utförligt, i många fall med bilder och källkod. Möjligheterna till att skapa Java-applets för att erbjuda ett fristående gränssnitt varifrån världen kan styras förklaras noggrant med ett körbart exempel. I SISUs VRML-applikation spelar databasaccesser en väsentlig roll. Olika tillvägagångssätt för att realisera databasaccesser i VRML-applikationer diskuteras, och det av SISU valda alternativet presenteras närmare.

Läsanvisning

- **Kap. 1** ***SISUs VRML-applikation***
  ger körinstruktioner och presenterar den funktionalitet som är inbyggd i applikationen.
- **Kap. 2** ***Realisering av funktionaliteten i VRML 2.0***
  presenterar utförligt hur applikationens funktionalitet är realiserad.
- **Kap. 3** ***VRML-utveckling***
  diskuterar kort det verktyg som använts samt alternativa verktyg.
- **Kap. 4** ***Slutsatser***
  sammanställer de slutsatser som kan dras efter att ha realiserat SISUs VRML-applikation.

[1] Virtual Reality Modeling Language. Genomgående syftas på version 2.0 av VRML om inget annat anges.

Innehåll

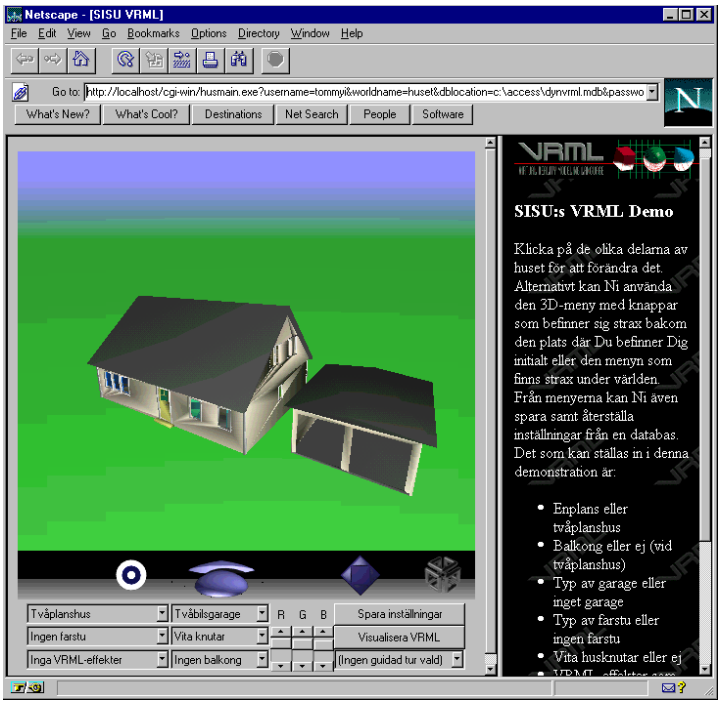
1. SISUs VRML-applikation

För att illustrera och utreda möjligheterna med VRML i interaktiva WWW-tillämpningar har SISU skapat en VRML-applikation som ligger till grund för denna rapport. Målet var att skapa dels en applikation som mycket väl skulle kunna vara en verklig applikation som något företag tagit fram, dels en applikation med hög grad av interaktion där själva VRML-världen genereras dynamiskt och där användaren ges möjlighet att läsa och spara information i en databas.

SISUs VRML-applikation utnyttjar merparten av den avancerade funktionalitet som VRML 2.0 erbjuder idag. VRML-världen i applikationen som beskrivs i denna rapport genereras dynamiskt och användaren ges möjlighet att lagra den aktuella konfigurationen av VRML-världen till databasen. Applikationen har en hög grad av interaktivitet och kan manipuleras direkt inuti världen. Förutom möjligheterna till interaktion inuti världen innehåller applikationen en från världen fristående Java-applet som kommunicerar med och kan styra världen. Dessutom finns ett antal ”guidade turer” som tar med användaren på automatiska rundvandringar i världen.

Valet föll på att skapa en applikation som en tillverkare av småhus skulle kunna tänkas stå bakom. Dessa tillverkare har ofta ett grundalternativ av en villa. Utifrån detta grundval kan spekulanten sedan välja just sin villa genom att välja till eller välja bort olika tillval som t ex garage och balkong. Detta har försökt efterliknas i applikationen. Förutom möjligheten att gå runt i världen och betrakta villan kan användaren välja till och välja bort komponenter interaktivt. T ex kan användaren välja vilken slags garage han vill ha, enplanshus eller tvåplanshus o s v. Fördelarna med applikationen är uppenbara. Spekulanten kan sitta hemma och i lugn och ro undersöka villan och dess alternativ från alla håll och kanter i den virtuella verkligheten.

När användaren är klar med sina val kan inställningarna sparas i en databas och när användaren/
spekulanten sedan kommer till den tänkta husfabrikanten finns redan alla önskemål angående tillägg, färger, m m lagrade i databasen och kan enkelt plockas fram. Detta skulle spara tid både för husfabrikanten och spekulanten. Dessutom har spekulanten fått möjlighet att närmare granska hur villan kommer att se ut då den är färdig vilket innebär att det är större sannolikhet att spekulanten blir nöjdare.



Figur A – En bild från SISUs VRML-applikation.

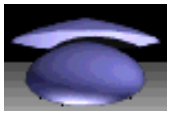
1.1 Körinstruktioner

Det är svårt att bilda sig en uppfattning om hur VRML fungerar utan att testa det själv. Därför finns SISUs applikation tillgänglig på Internet färdig att testköra. I dagsläget är den enda kombination av VRML-läsare och WWW-bläddrare, som stödjer all funktionalitet i SISUs applikation, CosmoPlayer tillsammans med Netscape 3.0 eller senare. Rekommenderad datorutrustning är minst en dator med Pentium processor samt 28800bps modem. Netscape 3.0 finns att hämta hem på http://www.netscape.com och CosmoPlayer VRML-läsare på http://webspace.sgi.com/cosmoplayer/download.html.

Navigering i CosmoPlayer

I CosmoPlayer finns det olika navigeringssätt. Den typ som är vald av SISUs applikation är ”walk”. I ”walk” läget finns det tre olika navigeringsknappar (symbolen allra längst till höger är Silicon Graphics logotyp). De olika knapparna är från vänster:

Rotation – Med den runda knappen längst till vänster som ska se ut som ett öga roterar man. Genom att klicka på denna knapp och hålla ned musknappen samt dra åt önskat håll roterar man åt det hållet.



Gå – Med den mittersta pil-liknande knappen ”går” man. Förflyttning sker genom att klicka och hålla ned musknappen samt dra åt det håll man vill gå. Med denna knapp är det även möjligt att rotera åt höger eller vänster. Drar man t ex snett uppåt åt höger sker en rotation åt höger samtidigt som man förflyttas framåt. Det är med denna knapp i princip all förflyttning lämpligen sker.

Förflytta vertikalt eller horisontellt – Den sista knappen används för förflyttningar, utan att rotera, uppåt, nedåt, vänster eller höger. Lämplig för att t ex ”ta en genväg” till övervåningen i huset genom taket.

Tyvärr stödjer den version av CosmoPlayer som finns tillgänglig när detta skrivs inte kollisionsdetektering. Hade CosmoPlayer stött kollisionsdetektering hade det varit betydligt enklare att navigera på ett verklighetstroget sätt. Nu är det möjligt att gå igenom väggar m m. Dessutom möjliggör kollisionsdetektering funktionalitet såsom att följa terrängen, t ex att gå upp och nedför trappan inuti huset. VRML 2.0 läsaren Community Place (tidigare kallad Cyber Passage) stödjer kollisionsdetektering och med den hade det varit möjligt att gå upp och ned för trapporna ”som i verkliga livet” om den läsaren hade stött all funktionalitet i SISUs VRML-värld.

Att använda applikationen
Applikationen är som sagt databasbaserad. Detta betyder att varje användare har sin egen profil med sina egna inställningar. Första gången man använder världen måste man följaktligen registrera sig i databasen för att få en egen profil. När registreringen är klar är det fritt fram att använda applikationen.

På http://dhcp6.sisu.se/huset finns det formulär varifrån man registrerar sig och sedan loggar på systemet. För närvarande finns bara en värld i databasen, men såsom applikationen är konstruerad är det möjligt att lägga till fler databasbaserade världar i samma databas allt eftersom.

När man loggar på *bör* man vänta tills **hela** dokumentet är inladdat innan man börjar använda systemet, d v s vänta tills det står ”Document: Done” längst ned i Netscapefönstret. Den vy som presenteras initialt är strax framför huset. Använd huvudsakligen den mittersta knappen (”Gå”-knappen) för att navigera. Strax bakom den initiala vyn finns även 3D-menyn.

Problem
Det har visat sig att CosmoPlayer kan vara svår att installera ibland. Om Live3D eller någon annan VRML-läsare dyker upp istället för CosmoPlayer då en VRML-värld läses in, trots att CosmoPlayer är installerad, så ladda ned och kör programmet Netscape Plug-in Chooser som finns på http://vs.spiw.com/vs/npc10.zip.

Den version av CosmoPlayer som finns tillgänglig idag (version 1.0 beta 2a) stödjer ej bakgrundsnoden (d v s himlens färger) om mindre än 16-bitars färgpalett är vald. Detta kommer antagligen också att åtgärdas i kommande versioner, men i dagsläget bör en färgpalett på minst 16 bitar väljas. För övrigt är version 1.0 beta2a som finns tillgänglig idag mycket instabil om en färgpalett med mindre än 16-bitars är vald.

P g a brister i CosmoPlayer måste man efter att ha valt en guidad tur välja StartView som viewpoint. Detta görs (på PC) genom att klicka med högra knappen och sedan välja Viewpoints->StartView.

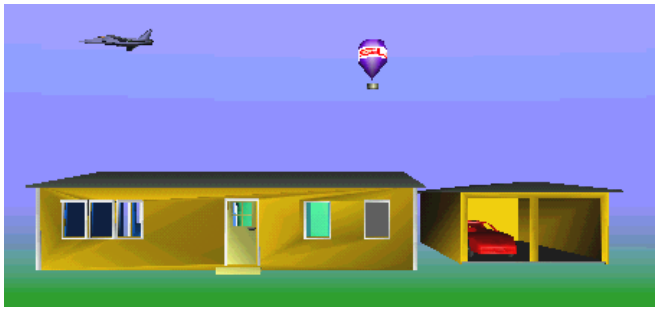
1.2 Funktionalitet i världen

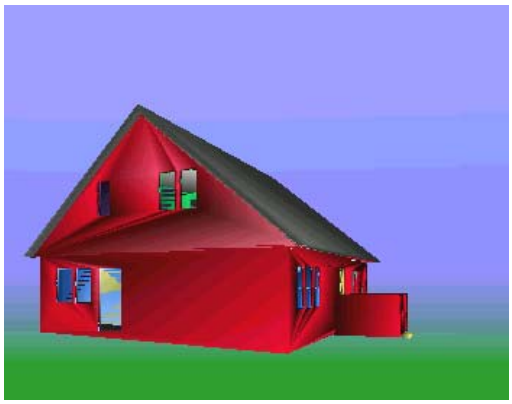
Är användarnamn och lösenord korrekta presenteras sedan en WWW-sida som är delad i två delar med hjälp av ”frames”. Den vänstra ramen innehåller VRML-världen och den högra

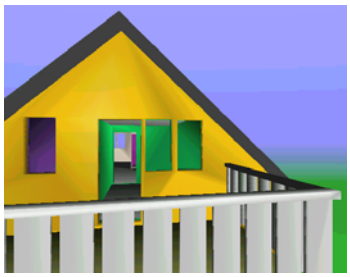
innehåller en mycket kort användarhandledning och information. Genom att navigera runt i VRML-världen och klicka på de olika delarna av huset kan användaren välja den variant av huset han vill ha.

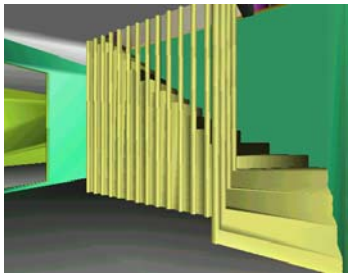
De olika delar av världen som går att ställa in är:

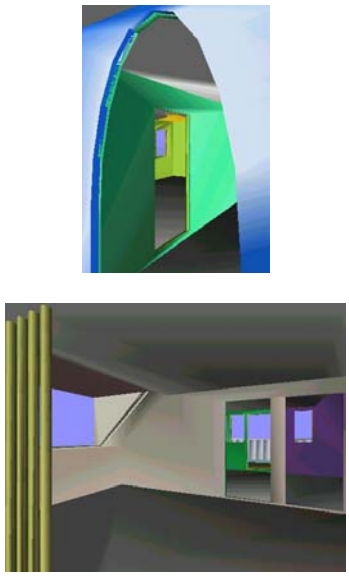
- **Garage** – Användaren kan välja mellan att ha carport, vanligt garage eller tvåbilsgarage. Valet görs genom att klicka på garaget.
- **Farstu** – Det är möjligt att välja ingen farstu alls (en liten dörrmatta), farstu utan tak eller farstu med tak. Denna inställning görs genom att klicka på farstun (eller dörrmattan).
- **Takhöjd** – Det finns tre olika tak. Genom att klicka på taket kan användaren välja mellan ett lågt tak (enplanshus) eller ett högt tak (tvåplanshus) med eller utan ett ”takfönster” som befinner sig strax ovanför farstun.
- **Balkong** – Om användaren väljer ett tvåplanshus erbjuds möjlighet att välja mellan att ha balkong eller ej. Finns en balkong synlig på vänstra sidan av huset väljs balkongalternativet bort genom att klicka på balkongen. Finns ingen balkong väljs balkongalternativet till genom att klicka på övervåningens vänstra husfasad (den plats där balkongen placeras).
- **Vita husknutar** – Tycker man det är trevligt med vita husknutar går det bra att klicka på bottenvåningens husfasad för att välja till eller välja bort detta alternativ.











Figur B – Bilder från olika varianter av huset.

Förutom de ovan nämnda möjligheterna till inställningar av världen behövs något mera. I många applikationer kan det tänkas att en eller flera knappar, eller en fullständig meny, behöver pre-senteras. Om någon slags meny ej används i SISUs applikation, hur ska det då vara möjligt för användaren att välja t ex att inte ha något garage alls? Detta val blir svårt utan knappar då garagevalet utan knappar sköts genom att nästa garageval hela tiden väljs då användaren klickar på det aktuella garagevalet. Men hur skall användaren kunna klicka på det aktuella valet ”inget garage alls” för att få nästa garagealternativ? (Ett liknande problem med farstun löstes med en dörrmatta som representerar valet ”ingen farstu alls”.)

I VRML 2.0 finns det två lösningar på detta problem. Bägge dessa har utvärderats och byggts in i applikationen. För det första är det fullt möjligt att bygga en egen variant av en meny *inuti* världen. I SISUs applikation har en 3D-meny byggts upp. Den är ett försök att representera knappar inuti världen. Problemet som uppstår för denna typ av objekt är: Hur ser en knapp ut i en tredimensionell rymd som försöker efterlikna verkligheten? I 3D-menyn är de valda alternativen alltid markerade på knapparna så att 3D-menyn hela tiden representerar de val som är aktiva i världen, oavsett om användaren väljer att modifiera världen genom att klicka på objekten eller genom att klicka på 3D-menyn. Fördelen med denna slags meny är att det är mycket enkelt att omforma den interaktivt allt eftersom världen förändras. Nackdelen är framförallt att det kan kännas lite onaturligt för användaren, framförallt eftersom resten av världen är relativt verklighetstrogen. För att förhöja användbarheten av 3D-menyn är den skapad så att den alltid är vänd mot användaren oavsett varifrån användaren betraktar den.

Figur C – 3D-menyn.

Den andra lösningen är utnyttja VRMLs externa gränssnitt för kommunikation med omvärlden. Det går att skapa en Java-applet som kan kommunicera med VRML-läsaren på ett mycket enkelt sätt. Exakt hur kommer att förklaras senare, men med hjälp av detta kan en mera traditionell meny skapas. Denna Javameny kommer att vara fristående från världen. Med gränssnittet är det möjligt att antingen låta en Java-applet styra världen, men även att låta världen styra en applet eller att låta världen och en applet styra varandra. SISUs Javameny är skapad så att den ser ut ungefär som ett traditionellt HTML-formulär. Alla förändringar i Javamenyn kan överföras dynamiskt till världen som omformas utan att någon annan del av världen påverkas och utan att dokumentet måste laddas om eller dylikt.

De val som finns tillgängliga från 3D-menyn och Javamenyn är förutom de tidigare nämnda valmöjligheterna:

- Om användaren valt ett tvåplanshus erbjuds möjlighet att via knapparna välja om balkong är önskvärt eller ej. Har användaren valt ett enplanshus omformas 3D-menyn dynamiskt så att detta alternativ ej är möjligt. Javamenyn omformas dock ej dynamiskt.
- ”VRML-effekter” är ett val som demonstrerar lite vad VRML enkelt kan åstadkomma. Detta har egentligen inget med ett hus att göra, utan är mera för att demonstrera VRMLs möjligheter. Väljs VRML-effekter presenteras en bil i garaget, en luftballong ovanför huset samt ett flygplan som flyger förbi med jämna mellanrum. Klickar användaren på bilen åker den iväg. Ballongen har SISUs logotyp mappad runt sig och roterar sakta runt ovanför huset. Den är även en länk: genom att klicka på ballongen kommer användaren till SISUs hemsida.
- Ett antal fördefinierade automatiska turer finns även tillgängliga från både 3D-menyn och Javamenyn. Dessa tar användaren med på en rundtur runt och/eller in i huset och sedan tillbaka. Dessa turer beror på hur världen för närvarande ser ut. ”Utvändig tur” låter användaren snurra ett varv runt huset och betrakta det utifrån från alla håll. ”Invändig tur nere” tar användaren med på en tur in i alla rum på bottenvåningen av huset och ”Invändig tur uppe” motsvarande på övervåningen. ”Invändig tur” är en tur både på bottenvåningen och övervåningen om en tvåplansvilla är vald. ”Fullständig tur” är en kombination av samtliga ovan nämnda och varar ca 3,5 minuter. Notera att turerna är medvetna om världens

tillstånd. Är balkongalternativet aktivt tar turerna med användaren ut på balkongen, annars gör den inte det o s v. Dessa turer går ej att avbryta så fort de väl är startade.

- Från Javamenyn finns även valmöjligheten att spara inställningar till databasen. (Anledning till att detta val ej finns i 3D-menyn är att den version av CosmoPlayer som finns tillgänglig idag saknar viss funktionalitet som krävs för detta.)
- Från 3D-menyn går det att återställa världen till det tillstånd som finns lagrat i databasen. Notera att om användaren väljer att återställa världen blir resultatet detsamma som att göra en ”reload” av sidan, d v s användaren kommer att placeras i den ursprungliga positionen.

Slutligen öppnas ytterdörren genom att man klickar på dörrhandtaget. Då förs handtaget först ned och sedan glider dörren upp samtidigt som dörrhandtaget sakta släpps upp.

2. Realisering av funktionaliteten i VRML 2.0

För en person som är van vid viss programmering är koden i VRML-världen extremt enkel. Den huvudsakliga funktionaliteten i denna applikation är Switchnoder och VRMLScript program som styr vilka alternativ i Switchnoderna som skall vara aktiva. Det enda som egentligen är lite krångligt i applikationen är att vissa alternativ måste synkroniseras med varandra så att det t ex inte finns någon balkong när användaren väljer ett enplanshus.

Valet att använda VRMLScript som scriptspråk beror inte på att VRMLScript är bättre än de andra tillgängliga scriptspråken, utan att CosmoPlayer för närvarande endast stödjer VRMLScript och att CosmoPlayer är den enda VRML-läsaren som erbjuder tillräckligt med funktionalitet för att realisera denna applikation. VRMLScript är en något modifierad och bantad variant av JavaScript. De övriga scriptspråk som skulle gå att använda är Java, som idag stödjs av Community Place (tidigare kallad Cyber Passage) och Liquid reality, samt JavaScript som kommer att stödjas av Live3D då den släpps i version 2.0.

2.1 Switch som medel för interaktivitet

Farstun och dess alternativ kan tas som exempel för att illustrera hur Switchnoder fungerar tillsammans med scripter. Farstualternativen kan sägas bestå av tre delar. Den första delen är själva farstun och dess olika alternativ. Den andra är knapparna i 3D-menyn för farstun. Slutligen har vi också scriptet som styr Switchnoderna.

Figur D – Närbild på ett av farstualternativen

Själva farstun består av grupperingsnoden Transform. En Transformnod har två funktioner: dels grupperar den ihop flera objekt till ett objekt, dels kan den utföra en geometrisk transformation med translation, rotation, skalningar, m m. Grupperingen av objekt gör att det är möjligt att t ex utföra geometriska transformationer på ett antal grupperade objekt som ett enda objekt och utan att förändra de objekt som befinner sig utanför denna gruppering. Varje gruppering får dessutom sitt eget *lokala* koordinatsystem som dessa geometriska transformationer utförs i.

```
DEF Farstu Transform {
  translation -1266 13556.6 2538.42
  children [
    DEF FarstuTouchSensor TouchSensor {}
    DEF FarstuSwitch Switch {
      whichChoice ##Farstu
      choice [
        DEF EndDmatta Transform {
          translation 5.65147 18.5731 2.16248
          children [
            ....
          ]
        },
        DEF UtanTak Transform {
          translation -4557.17 6.19226 -20.319
          children [
            ....
          ]
        },
        DEF MedTak Transform {
          translation -4560.91 8.56137 -28.464
          children [
            ....
          ]
        }
      ]
    }
  ]
}
```

Som synes består en farstu av en Transformnod och dess barn. Barnen är de olika objekten som är grupperade till att hanteras som ett objekt. I detta fall är barnen en TouchSensor som känner av ifall användaren klickar på något av de objekt som är grupperade under noden samt en Switchnod. En Switchnod har två olika fält. Det första är whichChoice som anger vilket av de olika alternativen i Switchnoden som skall vara aktivt, det andra är choice som är en lista med alternativen. Varje alternativ är i detta fall i sin tur en gruppering av noder som är de olika varianter av farstun som finns. Fältet whichChoice är i VRML 2.0 specifikationen deklarerat att vara ett ”exposedField” vilket innebär att man kan ändra dess värde dynamiskt och det är detta som utnyttjas för att låta användaren välja farstu interaktivt. Värdet för whichChoice (##Farstu) kan tills vidare ignoreras, det kommer att förklaras närmare i kapitlet om databaskopplingen i applikationen.

Den andra delen i applikationen som berör farstun är de knappar som finns i 3D-menyn:

```
Transform { # Första knappen – ingen farstu alls
  translation 0.522 0.156665 0
  children [
    DEF FarstuVal1Sensor TouchSensor {}
    DEF FarstuVal1 Switch {
      whichChild ##FarstuBox1
      choices [
        Shape {
          appearance Appearance {
            texture ImageTexture {
              url ["http://localhost/inlines/ejf.jpg"]
            }
          }
          geometry Box { size 0.26 0.146667 0.26 }
        },
        Shape {
          appearance Appearance {
            texture ImageTexture {
              url ["http://localhost/inlines/ejfX.jpg"]
            }
          }
          geometry Box { size 0.26 0.146667 0.26 }
        }
      ]
    }
  ]
}


Transform { # Andra knappen – farstu utan tak
  translation 0.522 0 0
  children [
    DEF FarstuVal2Sensor TouchSensor {}
    DEF FarstuVal2 Switch {
      whichChild ##FarstuBox2
      choices [
        ....
      ]
    }
  ]
}

Transform { # Tredje knappen – farstu med tak
  translation 0.522 -0.156665 0
  children [
    DEF FarstuVal3Sensor TouchSensor {}
    DEF FarstuVal3 Switch {
      whichChild ##FarstuBox3
      choices [
        ....
      ]
    }
  ]
}
```

Knapparna är alltså tre oberoende Transform grupperingsnoder. Alla tre har en Switch som väljer en av två olika aktiva knappar. Själva knappen är en Box som har en JPEG-bild mappad med knapptexten. Alla knappar har också en egen TouchSensor som känner av ifall användaren klickar på en viss knapp. Tredje delen, själva scriptet som styr Switchnoderna:

```
DEF FarstuScript Script {
 eventIn  SFTime  clicked
 eventIn  SFTime  click1
 eventIn  SFTime  click2
 eventIn  SFTime  click3
 eventOut SFInt32 val
 eventOut SFInt32 fval1
 eventOut SFInt32 fval2
 eventOut SFInt32 fval3

 url "vrmlscript:
  function initialize() {
   val = ##Farstu ;
   fval1 = ##FarstuBox1 ;
   fval2 = ##FarstuBox2 ;
   fval3 = ##FarstuBox3 ;
  }

  function clicked() {
   if (val == 2) {
    val = 0;
    fval1 = 1;
    fval2 = 0;
    fval3 = 0;
   }
   else {
    if (val == 1) {
     ....
    }
    else {
     ....
    }
   }
  }
  function click1() {
   val = 0;
   fval1 = 1;
   fval2 = 0;
   fval3 = 0;
  }
  function click2() {
   ....
  }
  function click3() {
   ....
  }
 "
}

ROUTE FarstuVal1Sensor.touchTime TO FarstuScript.click1
ROUTE FarstuVal2Sensor.touchTime TO FarstuScript.click2
ROUTE FarstuVal3Sensor.touchTime TO FarstuScript.click3
ROUTE FarstuScript.fval1 TO FarstuVal1.set_whichChoice
ROUTE FarstuScript.fval2 TO FarstuVal2.set_whichChoice
ROUTE FarstuScript.fval3 TO FarstuVal3.set_whichChoice
ROUTE FarstuTouchSensor.touchTime TO FarstuScript.clicked
ROUTE FarstuScript.val TO FarstuSwitch.set_whichChoice
```

Först i VRMLScript-rutinen deklareras olika variabler. Förutom de olika datatyperna en variabel kan ha (t ex heltal, flyttal, o s v) har varje variabel också en typ som avgör *hur* och *vad* som kan göras med variabeln, oavsett datatyp (jfr ”private”, ”public”, m fl i objektorienterade språk). De typerna som finns i VRML och VRMLScript är följande:

- **field** – En vanlig variabel som inte alls går att komma åt utifrån (jfr ”private”).
- **eventIn** – En variabel som kan *ta emot* meddelanden utifrån.
- **eventOut** – En variabel som kan *skicka* meddelanden till omvärlden.
- **exposedField** – En kombination av eventIn och eventOut som både kan skicka och ta emot meddelanden (jfr ”public”).

I VRMLScript-rutinen ovan syns att det finns ett antal variabler som tar emot meddelanden och ett antal som skickar meddelanden. När ett VRMLScript-program tar emot ett meddelande körs den funktion i programmet som har samma namn som variabeln som meddelandet skickades till.

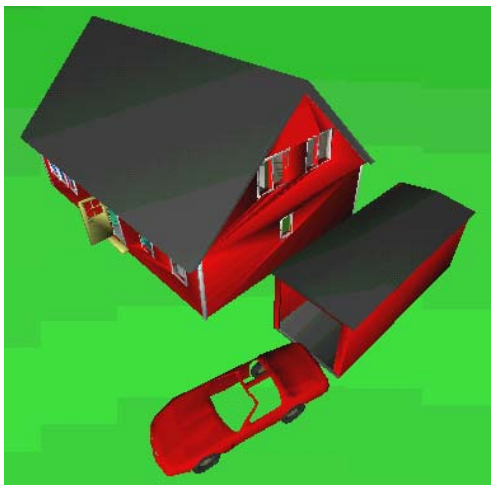
Tittar man efter själva scriptet finns ett antal ROUTEs. ROUTE är mekanismen som används i VRML 2.0 för att skicka meddelanden mellan olika delar av världen. De meddelanden som skickas är från olika klicksensorer som känner av ifall användaren klickar på t ex farstun i världen. Med-delanden från dessa klicksensorer skickas *till* programmets eventIn deklarerade variabler. Dessa meddelanden tas emot av programmet och värden för de eventOut deklarerade variablerna beräknas. Därefter skickas meddelanden *från scriptet* till de olika Switchnoderna. Dessa med-delanden innehåller information om vilket Switchalternativ som skall vara aktivt för tillfället. Modifieringen kan ske genom att t ex ROUTEa meddelanden till fältet.

Detta tillvägagångssätt möjliggör alltså interaktivitet genom att placera ut klicksensorer i världen. Då användaren klickar på en sådan sensor skickas ett meddelande till ett program som i sin tur beräknar nya meddelanden som skickas till olika ställen i världen. Förutom att skicka meddelanden för att uppdatera världen är det även möjligt att läsa av och skriva till de olika objekten i världen direkt från programmet (den funktionaliteten är ej implementerad i CosmoPlayer 1.0 beta 2).

Detta är ett VRMLScript program, men det skulle lika gärna kunna vara ett program i något annat språk. Idag finns det läsare som stödjer VRMLScript, JavaScript och Java, men det finns inget hinder för att skapa en läsare med stöd för något annat scriptspråk.

2.2 Interpolatorer för rörelse

Nästa sak som kan vara intressant att studera är hur de olika rörelser som finns i världen fungerar. Som exempel används bilen som åker iväg när användaren klickar på den:



Figur E – Bilen som åker iväg.

```
DEF Bil Transform {
  translation 8.3 0.7 -16
  scale 0.4 0.4 0.4
  rotation 0 1 0 -0.48
  children [
    DEF BilTouchSensor TouchSensor {}
    DEF BilTime TimeSensor {
      cycleInterval 35
    }
    DEF BilPosition PositionInterpolator {
      key [0 0.2 1]
      keyValue [8.3 0.7 -16,
                8.3 0.7 -10,
                -50 0.7 70]
    }
    DEF BilRotation OrientationInterpolator {
      key [0 0.1 0.3 1]
      keyValue [0 1 0 -0.48,
                0 1 0 -0.48,
                0 1 0 -1.3,
                0 1 0 -1.3]
    }
    Inline {
     url ["http://localhost/objects/corvette2.wrl"]
    }
  ]
}
```

Bilen består även den av grupperingsnoden Transform som placerar bilen på korrekt plats i garaget vid start. I grupperingen finns en klicksensor som känner av om användaren klickar på bilen. Det nya i detta exempel är tidssensorn och de två interpolatorerna. TimeSensor är egent-
ligen ingen sensor, utan snarare en generator. Den används för att generera ”ticks” i en viss tid. Tidssensorns fält cycleInterval anger hur lång tid den ska generera ticks, i detta fall är fältet
satt till 35 sekunder. PositionInterpolator används för att interpolera XYZ-värden. Fälten i PositionInterpolator är keys som anger den relativa tiden mellan de olika positionerna samt values som anger positionerna. T ex innebär det första steget i positionsinterpolatorn att positionerna mellan XYZ = 8.3 0.7 -16 och XYZ = 8.3 0.7 -10 interpoleras under 20% av den totala inter-polationstiden som i detta fall är 0.2 * 35 = 7 sekunder.

OrientationInterpolator fungerar på ett motsvarande sätt, men den interpolerar rotationer. En rotation består av XYZ-värden som anger runt vilken axel rotationen skall ske samt ett värde som anger hur många radianer objektet skall roteras. Exempelvis skall rotation ske runt Y-axeln (XYZ = 0 1 0) från -0.48 radianer till -1.3 radianer vid tiden 10% till 30% av totala interpolationstiden (7 sekunder i detta fall).

Själva bilen finns i en separat fil och är inkluderad i världen som en Inlinenod. Inline betyder att VRML-läsaren läser in filen först när den behövs. Har användaren valt ”Inga VRML-effekter” i 3D-menyn kommer bl a bilen inte att synas och alltså inte läsas in av VRML-läsaren överhuvud-taget. När användaren sedan aktiverar VRML-effekterna kommer bilen att läsas in och visas i världen. Värt att notera angående bilen kan också vara att den är hämtad från ett av de bibliotek av VRML-objekt som redan börjat dyka upp.

```
ROUTE BilTouchSensor.touchTime TO BilTime.startTime
ROUTE BilTime.fraction_changed TO BilPosition.set_fraction
ROUTE BilTime.fraction_changed TO BilRotation.set_fraction
ROUTE BilPosition.value_changed TO Bil.translation
ROUTE BilRotation.value_changed TO Bil.rotation
```

Slutligen finns det ett antal ROUTEs som vidarebefordrar meddelanden. De meddelande som skickas är i detta fall ett meddelande från klicksensor BilTouchSensor till tidssensorn BilTime
då användaren klickar på bilen. De delar av tiden som kontinuerligt genereras under de kommande 35 sekunderna vidarebefordras till de bägge interpolatorerna BilPosition och BilRotation. Med-delandet tas emot av det fält i interpolatorerna som anger hur långt i tidscykeln tidssensorn kommit. Interpolatorerna genererar i sin tur value_changed som är positions- respektive rotations-värden för den tid som togs emot från tidssensorn. Dessa värden vidarebefordras slutligen till translation- och rotationfälten i bilens Transform grupperingsnod och som gör resultatet synligt för användaren. D v s, bilen kommer att både förflyttas och roteras *samtidigt.* Då alla translationer och rotationer hela tiden sker i objektets *lokala* koordinatsystem kommer det att bli en mjuk svängrörelse och bilen rör sig i en liten cirkelbåge under rotationen. Vill man ha en ännu mera verklighetstrogen rörelse kan man dels ange fler rotationer och positioner i mindre steg. Dels kan man ange ett annat rotationscentrum än mitten på objektet. Det är möjligt att rotera runt t ex en punkt som befinner sig mellan bakhjulen vilket troligen skulle likna en verklig bils rörelse ännu mer.

De guidade turerna har realiserats på ett liknande sätt som bilens rörelse. Guidade turer är när användaren tas med på en datorstyrd tur runt världen. Det finns två alternativ med vars hjälp detta är realiserbart. Det första är att manipulera med användarens ”kamera” (ViewPointnoden) för att transportera användaren genom världen. Det andra är att låta ”kameran” vara fast och istället flytta runt hela världen. Bägge har sina fördelar. Att manipulera kameran är *verkar* mera intuitivt och lättförståeligt. Men i verkligheten fungerar det inte riktigt som man tänkt sig eftersom man då hela tiden använder kamerans lokala koordinatsystem. Detta innebär att det kan vara svårt att beskriva en rörelse mellan exakta punkter i *världens* koordinatsystem då det lokala koordinatsystemet kommer att förändras vid rotationer. Det tillvägagångssätt som istället valdes var att gruppera *hela världen med samtliga objekt* i två stycken Transformnoder för att kunna utföra turerna i världens koordinatsystem:

```
DEF HelaScenenRotation Transform {
 children [
  DEF HelaScenenTranslation Transform {
   children [
    ....
   ]
  }
 ]
}
```

Med detta tillvägagångssätt för guidade turer sker rotationer och translationer mer intuitivt som man tänker sig att de borde fungera. Rotationer roterar hela världen och translationer förflyttar hela världen på ett sådant sätt att man kan använda det ursprungliga ej manipulerade världs-koordinatsystemet för translationerna. För själva turerna anges liksom för bilens rörelse ett antal punkter i XYZ-rymden, rotationer och en tidsaxel och sedan koordineras dessa ihop till en lämplig rörelse.

Förutom de ovan nämnda interpoleringsnoderna finns det även möjlighet att interpolera färger, skalor på objekt, former, m m. Interpolation av former innebär att man omformar ett objekt till

ett annat objekt med samma antal punkter i en mjuk rörelse, vilket är en mycket kraftfull funk-tionalitet.

2.3 Återanvändning med prototyper och instansieringar

Instansieringar och prototyper är en mycket kraftull mekanism för att ge möjlighet att återanvända kod. Som synts har DEF använts relativt flitigt. DEF tjänar två syften, dels blir koden mer läsbar om man namnger objekt, dels instansieras objekt och kan sedan återanvändas. Vill man återanvända en nod som är instansierad med ”DEF någotnamn” skriver man helt enkelt ”USE någotnamn”. Prototyper har tyvärr inte kunnat användas p g a brister i CosmoPlayer. Men protoyper anges som en av de mest kraftfulla mekanismerna i VRML 2.0. Tanken var att alla små boxar i 3D-menyn skulle skapas med prototyper, men det fungerar inte riktigt som det ska idag. En prototyp för dessa boxar skulle se ut ungefär som:

```
PROTO MenyBox [
 exposedField MFString urlField  [""]
 exposedField MFString urlFieldX [""]
 field        SFVec3f  boxSize   0 0 0
 exposedField SFVec3f  pos       0 0 0
 exposedField SFInt32  select    0
 ]
{
 Transform {
  translation IS pos
  children [
   Shape {
    appearance Appearance {
     texture Switch {
      whichChoice IS select
      choice [
       ImageTexture {
        url IS urlField
       },
       ImageTexture {
        url IS urlFieldX
       }
      ]
     }
    }
    geometry Box {
     size IS boxSize
    }
   }
  ]
 }
}
```

Denna prototyp ser ganska komplex ut, men principen är enkel. Man anger en prototyp med ”PROTO prototypnamn” följt av olika parametrar. I själva kroppen till prototypen refererar man sedan till dessa parametrar med IS. Anrop av prototypen sker sedan med ”prototypnamn { parameternamn1 parametervärde1 parameternamn2 parametervärde2 etc }”. Förutom denna möjlighet till återanvändning finns även nodtypen EXTERNPROTO som fungerar på samma sätt som PROTO med skillnaden att implementationen av prototypen finns någon annanstans på Internet och kroppen av prototypen är följaktligen en (eller flera) URL/URN-länkar. Detta innebär att det räcker med att en enda person på Internet skapar en komplex prototyp, och sedan kan samtliga personer återanvända den genom att ange URL/URN till prototypen i en EXTERNPROTO. Tanken är att EXTERNPROTO tillsammans med Script ska kunna förverkliga de flesta av de framtida utbyggnader av VRML-standarden som kommer att uppstå.

2.4 Övrig funktionalitet i språket VRML 2.0

Som tidigare nämnts är 3D-menyn alltid riktad ungefär mot användaren. Den är ihopgrupperad med en s k Billboardnod. Billboardnoder ser till dess lokala Z-axel alltid pekar åt samma håll som användaren tittar. D v s användaren får bästa möjliga översikt av alla knappar på menyn oavsett vilken riktning han betraktar den från. I VRML ser detta ut som följer:

```
Billboard {
 axisOfRotation 0 0 0
 children [
  ...
 ]
}
```

Klickar användaren på luftballongen, som har SISUs logotyp mappad runt sig, kommer användaren att tas till SISUs hemsida. Mappningen av logotypen är mycket enkel:

```
DEF BallongInline Transform {
 ....
 children [
  DEF BallongTexture Transform {
   ....
   children [
    Shape {
     appearance Appearance {
      texture ImageTexture {
       url ["http://www.sisu.se/jobs/sisu.gif"]
       repeatS FALSE
       repeatT FALSE
      }
     }
     geometry Cylinder { radius 760 height 992.608 }
    }
   ]
  }
  ....
 ]
}
```

Det som är utklippt ovan är den del av ballongen som är cylindern med SISU-logotypen. I appearancefältet anges normalt färg m m, men används texturefältet är det bara att tillhanda-hålla en URL till den bild som skall mappas samt ange om den ska repeteras i X och/eller Y-led eller om den ska sträckas ut till hela formens storlek. Är det inte tillräckligt med att mappa en bild på ett objekt erbjuds i VRML även möjligheten att mappa filformat som MPEG på objekt. MPEG är ett filformat som används för t ex digitala versioner av filmer. Detta skulle t ex kunna användas till en TV om man skulle ge användaren möjlighet att möblera huset. WWW-länken anges också mycket enkelt:

```
Anchor {
 parameter  ["target=_top"]
 description "SISUs hemsida"
 url ["http://www.sisu.se"]
 children [
  Inline {
   url ["http://localhost/inlines/ballong.wrl"]
  }
  ....
 ]
}
```

Anchornoden är en grupperingsnod liknande Transform. Skillnaden är att istället för att ange geometriska transformationer anges en URL med parametrar. Att notera är att själva ballongen är inkluderade med en Inlinenod.

För att skapa en realistisk omgivning runt huset har en Backgroundnod använts. Färger är definie-rade till himlen på ett sådant sätt att det ska ge en relativt realistisk bakgrund. Tillvägagångssättet i VRML är att en oändligt stor sfär som alltid befinner sig oändligt långt från användaren omsluter världen. För att färga himlen anges vinklar. Den första vinkeln är noll och den anges inte. Vinkel noll innebär att den punkten befinner sig rakt ovanför användaren. Sedan anges vinklar i radianer från denna punkt. Lämpliga färger tilldelas de olika punkterna, och färgerna interpoleras för en mjuk övergång mellan punkterna.

```
Background {
 skyAngle [0.9, 1.56]
 skyColor [0.7 0.7 1, 0.55 0.55 1, 0.4 0.4 1]
}
```

Här är alltså vinklarna 0, 0.9, samt 1.56 radianer angivna. De färger som anges är olika blåa färger (RGB-värden) som interpoleras mellan vinklarna för att ge en någorlunda realistisk himmel. Förutom möjligheten att skapa himmel med Backgroundnoden kommer det så småningom vara möjligt att skapa en ännu mera verklighetstrogen bakgrund. Mark kan anges på ett liknande sätt som himlen, berg som befinner sig ”i horisonten” kan skapas med texturer och det ska bli möjligt att skapa en dimma som både kan används för att skapa en verklighetstrogen dimma och för att styra dämpning av färger m.m. på avlägsna objekt.

Förutom den i applikationen inkluderade funktionaliteten finns även följande funktionalitet i VRML 2.0:

- **Detaljnivå** – LOD (Level Of Detail) är en nod som anger ett alternativt objekt då objektet befinner sig långt borta från användaren. T ex skulle flygplanet ha kunnat ha en mycket låg detaljnivå då det befinner sig lång bort från användaren, vilket skulle spara beräkningar för datorn.
- **Ljud** – I VRML finns möjligheten till 3D-ljud. T ex hade bilen kunnat ha ljud som rör sig med bilen när den åker iväg och kanske en tutsignal om användaren kommer för nära bilen när den kör iväg.
- **Ljus –** I applikationen anges endast fasta ljuskällor, men man skulle kunna tänka sig att ha t ex en sol som går upp och ner i bakgrunden som alstrar ljus, eller att bilens strålkastare lyser upp framför sig.
- **Sensorer** – De enda sensorer som används är sensorer som genererar ”ticks” för alla animationer och rörelser samt sensorer som känner av musklick. Sensorer som känner av

att muspekaren befinner sig ovanför ett objekt skulle kunna användas till t ex att visa en informativ text (med textnoden som heller ej använts) då muspekaren befinner sig ovanför ett objekt man kan klicka på. Det finns en sensor som känner av om man befinner sig inom ett visst område som t ex kan användas till att uppdatera information i en frame om vilket rum man befinner sig i eller för att öppna ytterdörren då man närmar sig den. En annan sensor känner av ifall ett objekt är synligt eller ej. Det finns även sensorer för att låta användaren manipulera världen genom att förflytta objekt i något plan eller för att rotera objekt runt någon axel. Dessa skulle kunna användas till att t ex ge användaren möjlighet att pröva olika möbleringar i huset eller kanske flytta någon vägg.

2.5 Externt Javagränssnitt

En funktionalitet som möjliggör skapandet av mycket kraftfulla VRML-applikationer är det externa API-gränssnitt som finns för att kommunicera med omvärlden. Med hjälp av Netscapes LiveConnect är det möjligt att låta VRML-världar kommunicera med externa Java och till viss del även JavaScript program. Det bästa sättet att förklara hur det externa Javagränssnittet fungerar är med ett exempel. I SISUs applikation finns det en Java-applet som kan styra det mesta i VRML-världen, men vi begränsar oss till en applet som enbart styr husets färg i detta exempel. Det som behövs för detta är tre delar:

1. VRML-noden som skall refereras måste ges ett namn med DEF
2. I Javaprogrammet måste man, via LiveConnect och JavaScript, få en referens till VRML-läsaren och världen.
3. Från Javaprogrammet kan man sedan komma åt noden med kommandot getNode()

I detta exempel skapar vi ett HTML-dokument med världen inkluderad med HTMLs embed kommando:

```
<html>
<head><title>VRML-Java exempel</title>
</head>
<body>
<center>
<embed src="http://localhost/huset_tommyi.wrl" width=100% height=85%
name="test_world">
<applet codebase="/java/Husdemo" code="ChangeColor.class" width=450 height=70
mayscript>
</applet>
</center>
</body>
</html>
```

Inga konstigheter än så länge. Det enda är att parametern ”mayscript” måste ges som parameter till <applet ...> för att låta Java kommunicera via LiveConnect.

I applikationen ser det ut som följer i VRML-filen:

```
Shape {
 appearance Appearance {
  material DEF HusFasadMaterial Material { diffuseColor 1.0 0.0 0.0 }
 }
 geometry .....
 ...
}
```

Vad vi vill komma åt är färgen på huset, den finns i en Materialnod som är instansierad och given namnet HusFasadMaterial med DEF. Notera att denna instansiering innebär att det är möjligt att återanvända HusFasadMaterial till andra delar av applikationen. Exempelvis kommer även garagets färg att förändras då husets färg förändras eftersom garaget återanvänder det instansierade objektet HusFasadMaterial. Färgen är angiven i RGB-systemet och har tre komponenter mellan noll och ett för att bestämma färgens halt av röd, grön samt blå.

Den externa Java-applet som styr husets färger i sin helhet:

```
import java.awt.*;
import java.applet.Applet;
import vrml.Browser;
import vrml.Node;
import vrml.field.*;
import netscape.javascript.JSObject;

public class ChangeColor extends Applet {
 private Browser browse;
 private Scrollbar rScroll;
 private Scrollbar gScroll;
 private Scrollbar bScroll;

 public void init() {
  setLayout(new FlowLayout());
  add(new Label("Röd:",Label.RIGHT));
  rScroll = new Scrollbar(Scrollbar.HORIZONTAL,24,0,0,25);
  add(rScroll);
  add(new Label("Grön:",Label.RIGHT));
  gScroll = new Scrollbar(Scrollbar.HORIZONTAL,0,0,0,25);
  add(gScroll);
  add(new Label("Blå:",Label.RIGHT));
  bScroll = new Scrollbar(Scrollbar.HORIZONTAL,0,0,0,25);
  add(bScroll);

  JSObject win = JSObject.getWindow(this);
  browse = (Browser)win.eval("document.test_world");
 }

 public boolean handleEvent (Event evt) {
  if(evt.target instanceof Scrollbar) {
   Node n = browse.getNode("HusFasadMaterial");
   EventInSFColor col = (EventInSFColor)n.getEventIn("set_diffuseColor");
   float value[] = new float[3];
   value[0] = (float)rScroll.getValue()/(float)25.0;
   value[1] = (float)gScroll.getValue()/(float)25.0;
   value[2] = (float)bScroll.getValue()/(float)25.0;
   col.setValue(value);
  }
  return true;
 }
}
```

Detta exempel förutsätter viss Java- och JavaScript-kunskap för full förståelse. I init() i Javakoden skapas dels själva ”scrollbar”-knapparna m m som klassvariabler, dels hämtas en referens till VRML-läsaren och världen. Variabeln ”win” är deklarerad som ett JSObject, vilket är ett JavaScript objekt. JavaScript har tillgång till och kan referera till allt inuti Netscape och med JSObject.getWindow(this) hämtar vi en JSObject referens till hela Netscape fönstret. I Netscape fönstret finns även VRML-filen inkluderad med HTML-kommandot EMBED och given namnet test_world. Evalueras nu JavaScript kommandot "document.test_world" fås en pluginreferens till objektet i dokumentet vid namn test_world. Resultatet blir alltså att LiveConnect nu har gett variabeln browse en referens till VRML-

världen/läsaren och att det är fritt fram att kommunicera! Detta tillvägagångssätt för att få en referens till VRML-världen/läsaren baserar sig på LiveConnect och ingår inte i specifikationen för externt API. Om det i framtiden kommer andra liknande lösningar för att få referenser till VRML från en applet kommer det (antagligen) att fungera lika bra att använda dem.

I handleEvent, som bl a anropas då användaren ändrar på rullningslisterna, sker nu kommunika-tionen. Med getNode("HusFasadMaterial") hämtas en referens till noden HusFasadMaterial. Det fält vi vill komma åt är fältet set_diffuseColor som är av typen EventIn SFColor. Vi skapar då ett objekt av klassen EventInSFColor som är den VRML-specifika Javaklass som hanterar denna VRML-typ. Med getEventIn("set_diffuseColor") hämtas en referens till just det fältet. SFColor är egentligen tre stycken flyttal, så genom att skapa en array med tre flyttal kan man sedan uppdatera det fältet med setValue().

På ett liknande sätt kan man även läsa av värden i VRML-världen med getValue(). Med klassen EventOutObserver kan man även skapa en metod callback() som kan kopplas till en specifik eventOut i VRML-världen och anropas så fort ett meddelande skickas från eventOut-fältet, d v s skicka meddelanden från VRML-världen till Javaprogrammet!

Notera att detta tillvägagångssätt för att kommunicera mellan Java och VRML är Silicon Graphics förslag till externt API som redan är delvis implementerat i CosmoPlayer. Andra förslag existerar, men som det ser ut för tillfället verkar det som om detta förslag kommer att inkluderas i VRML-standarden för kommunikation med omvärlden.

(Värt att veta: I nuvarande version av specifikationen för externt API står det set() och get() istället för setValue() och getValue(), så det kan hända att den version av CosmoPlayer Ni använder inte känner igen setValue() och getValue() utan vill ha set() och get() istället.)

2.6 Databasaccess

Ett av de primära syftena med VRML-projektet var att undersöka möjligheterna att skapa interaktiva databasbaserade sytem med VRML som användargränssnitt. De huvudsakliga alternativa tillvägagångssätten är att skapa ett program som genererar hela VRML-världen på ett intelligent sätt, att spara hela VRML-världen i databasen, eller att ha en vanlig VRML-källkodsfil med databasvariabler i.

Dessa alternativ har vägts mot varandra och har alla sina fördelar och nackdelar. Det mest omständliga tillvägagångssättet är det första alternativet, att ha ett program som genererar hela VRML-världen i princip helt självständigt. Detta kräver ganska mycket intelligens från programmet. Ett något jämförligt problem kan vara att studera de problem som ligger i att ut-veckla en utvecklingsmiljö för VRML 2.0. Att skapa en editor eller liknande som klarar av samtlig funktionalitet i VRML 2.0 är avancerat, och att skapa ett CGI (eller liknande) som genererar kod helt självständigt är ett nästan jämförbart svårt problem. Undantaget är vissa applikationer eller delar av applikationer som t ex tredimensionella grafer eller matematiska samband. Dessa applikationer är ofta svåra att uttrycka på ett någorlunda statiskt sätt, utan VRML-källkoden måste helt enkelt genereras och beräknas fram. Men dessa exempel på applikationer behöver ej bli komplexa ändå eftersom de endast berör en mycket liten delmängd av hela funktionaliteten i VRML.

Alternativet att spara VRML-världen i sin helhet i databasen har sin fördel i andra applikationer. I detta fall är det viktigare att lägga ”intelligensen” i systemet på en bra struktur i databasen och sedan ha ett program som bara plockar ut delar av databasen och sammanställer dessa till en VRML-värld. En variant av SISUs VRML-applikation hade kunnat bli lämplig för detta. Antag att det ej skulle vara möjligt för användaren att omforma huset interaktivt, d v s det skulle ej gå att växla mellan de olika alternativen av huset direkt i VRML-världen. Ett formulär där användaren matar in de parametrar han vill ha för sitt hus presenteras. Ett CGI slår upp dessa val i en databas och för varje val plockas *ett* alternativ ut ur databasen och läggs in i den genererade filen. T ex skulle det inte finnas en Switchnod som reglerade vilket av farstualternativen som är aktivt för tillfället, utan det skulle bara finnas ett farstualternativ. Resultatet skulle bli en betydligt mindre fil eftersom alla alternativen ej behöver överföras till användaren. Dock skulle världen antagligen ej besitta samma möjligheter till interaktion.

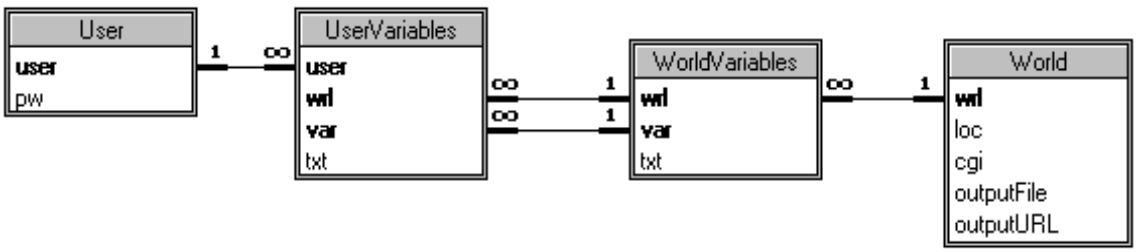
Men i SISUs VRML-applikation var strävan att hålla möjligheterna till interaktion på en hög nivå samtidigt som de ovan nämnda fallen ej var önskvärda. Användaren av applikationen skulle kunna omforma världen interaktivt och läsa samt spara sin världskonfiguration till databasen. Det alter-nativ som låg närmast till hands var då det senaste alternativet där världadminstratören skapar en VRML-fil som vanligt och inför variabler i den som sedan slås upp i en databas, samt att variab-lerna även skall kunna hämtas från världen och lagras i databasen.

Vad som behövs för detta är notation för variabler, en lämplig representation i en databas, samt ett eller flera CGI-program som ersätter variablerna med dess lagrade värde.

I SISUs VRML-applikation är själva utseendet på VRML-källkodsfilen identiskt med en vanlig VRML-fil, med tillägg för databasvariabler. Dessa databasvariabler identifieras i applikationen genom att de har ”##” som prefix.

```
DEF FarstuSwitch Switch {
  whichChild ##Farstu
  choices [
    ...
```

I exemplet ovan syns en Switchnod där databasvariabeln Farstu anger vilket av de olika farstu-alternativen som är aktivt när användaren laddar in världen. Kriterier för databas-representationen är framförallt att varje användare måste kunna lagra sina variabelvärden oberoende av andra användare samt att om användaren ej har några värden lagrade måste databasvariablerna ersättas med några fördefinierade standardvärden. Förutom detta kan det vara önskvärt att lagra lösenord eller liknande för att identifiera användare samt diverse kringinformation om världen som kan behövas.



Figur 6 – Databasrepresentation, nycklar med fetstil.

En kort beskrivning av databastabellerna:

- **User** – Lagrar användarnamn samt lösenord.
- **UserVariables** – Lagrar en viss användares värde för en databasvariabel.
- **WorldVariables** – Lagrar standardvärden för en databasvariabel.
- **World** – Lagrar diverse information om världen.

För att dynamiskt generera VRML-världen i applikationen används olika CGIer. Arbetsgången när användaren ”loggar på” systemet är som följer i SISUs applikation:

Ett HTML-**formulär** med fält där användaren matar in namn, lösenord samt önskad värld presenteras.

Ett **huvud-CGI** tar emot data från formuläret och kontrollerar användarnamn och lösenord mot databasen. Om det stämmer slår huvud-CGIet upp värdet på fältet cgi i tabellen World. Detta fält innehåller det CGI som skall användas för att genereras just denna värld, kallat för applikations-CGI. Huvud-CGIet returnerar denna information till användarens WWW-bläddrare som kommer att anropa applikations-CGIet istället. Detta tillvägagångssätt innebär att samtliga världar kan lagras i en och samma databas samtidigt som varje värld har sin alldeles egna CGI som hanterar just den världen. Givetvis *måste* inte ett nytt skräddarsytt CGI skrivas för varje ny värld, i SISUs applikation ingår även ett CGI för det generella fallet som kan tillhandahållas som applikations-CGI.

Applikations-CGIet kontrollerar även det användarnamn och lösenord och börjar sedan generering av världen. I SISUs applikation skapas två frames m m och själva världen kommer att genereras senare i en av dessa. En generell rutin för att ersätta variabler kontrollerar innehållet i fältet loc i tabellen World. Det fältet innehåller fysisk sökväg till VRML-källkodsfilen. Den generella rutinen för översättningen av databasvariablerna kan generera världen på två sätt, det första är att den presenterar världen direkt för användaren genom att skicka filen rad för rad till användaren, det andra är att den kan lagra filen på disk (sökväg finns i outputFile) och att världen sedan presenteras för användaren genom att använda fältet outputURL som är URL till den genererade filen. Det den generella genereringsrutinen gör är att den läser in VRML-källkoden rad för rad. Hittar den en variabel ersätts den med dess värde. Sökning efter variabelns värde sker i följande ordning:

1. Om det finns en CGI-parameter ”variabelnamn=något” ersätts variabeln med detta värde. Detta innebär att det är möjligt att direkt från ett formulär eller ett annat CGI göra inställningar som har högre prioritet än de lagrade värdena.
2. Om användaren har denna variabel lagrad ersätts variabeln med detta värde.
3. Annars ersätts variabeln med dess standardvärde (eller felmeddelande om standardvärde ej existerar).

Att låta användaren lagra databasvariablernas värden ställer ett par krav. För det första måste användarnamn m m finnas tillgängligt. För det andra måste alla, eller åtminstone vissa, databasvariablers aktuella värden finnas tillgängliga. Är det önskvärt att erbjuda möjlighet att lagra databasvariabler från inuti världen behövs ett script dit alla variablers värden vidarebefordras eller finns tillgängliga. Från detta script kan ett CGI med lämpliga parametrar startas eller så kan en Anchornods URL-länk modifieras från scriptet så att den kontinuerligt

innehåller korrekta parametrar för databaslagring via ett CGI. Alternativt kan möjligheten att lagra databasvariablerna läggas i ett program som är fristående från världen.

Det alternativ som valts i SISUs VRML-applikation är det enda som är realiserbart idag då de andra två kräver funktionalitet som ännu inte är implementerade i dagens version av CosmoPlayer. I SISUs applikation läser en extern Java-applet av samtliga variabler i världen då de skall lagras, varpå ett CGI med alla variablers värden anropas för att utföra själva lagringen.

3. VRML-utveckling

Hur går man då tillväga för att skapa sin egen VRML-baserade applikation? Det första som bör konstateras är vilken typ av värld som är önskvärd. Är det många komplexa objekt eller är det huvudsakligen enklare former? Anledningen till att det bör beaktas är att utvecklingsmiljöerna ofta är inriktade mot en speciell typ av världar.

I SISUs VRML-applikation gjordes bedömningen att formerna i världen var ganska enkla. Komplexiteten låg i de program m m som skulle styra världen. Paragraph har två produkter som är *mycket* enkla att använda men som endast är lämpade för enklare former. Dessa produkter är Virtual Home Space Builder (VHSB) som är inriktad på att enbart bygga med klossar, samt Internet 3D Space Builder (ISB) som liknar VHSB men även klarar andra former än klossar. ISB fanns under utvecklandet av SISUs applikation endast i en tidig betaversion, men valet föll på den i alla fall. Med ISB byggs världar helt med musen och ett ”drag-and-drop” gränssnitt. Det tar bara ett par minuter att lära sig att behärska programmet. Nackdelen är som sagt att komplexare former, som t ex bilen i applikationen, ej går att modellera i ISB. Är det bara ett eller ett fåtal komplexa objekt kan dessa modelleras i en ”vanlig” 3D-utvecklings-miljö och sedan konverteras till VRML och inkluderas manuellt i världen. Men baserar sig stora delar av världen på komplexare objekt bör kanske någon annan utvecklingsmiljö beaktas.

Den allra mest avancerade utvecklingsmiljön, som de flesta professionella skaparna av VRML-världar i USA (se t ex http://www.planet9.com) använder, är Cosmo Worlds (tidigare kallat Cosmo Create 3D) för Silicon Graphics maskiner. Tyvärr har SISU ej fått någon förhandstitt
på Cosmo Worlds, men de som använder den säger att Cosmo Worlds ligger minst ett år före i utvecklingen än samtliga andra VRML-utvecklingsmiljöer. Miljön i Cosmo Worlds är även den helt grafiskt och kodfri (förutom scriptdelen).

Är det inte önskvärt att investera i en Silicon Graphics maskin finns det ett par andra något enklare och mera begränsade utvecklingsmiljöer för PC som kan vara värda att titta på. Pioneer
har ett ganska speciellt användargränssnitt men kan vara mycket lämpat för att bygga komplexa objekt då man behärskar användargränssnittet. Virtus Walkthrough Pro är relativt lättanvänd samt kan skapa ganska komplexa världar och kan även det vara ett alternativ. Bägge dessa har dock i dagsläget endast stöd för VRML 1.0.

I SISUs applikation är relativt mycket gjort för hand i en texteditor, och investerar man inte i en Silicon Graphics dator får man nog räkna med att bearbeta VRML-världarna manuellt till stor del. I och med att VRML är allmänt accepterad av Internetaktörerna är ett större utbud av utveck-lingsmiljöer att räkna med i framtiden. Förutom de rena VRML-utvecklingsmiljöerna kommer säkerligen de flesta existerande 3D-utvecklingsmiljöer i framtiden att bygga in stöd för VRML 2.0. För utförligare dokumentation om utvecklingsmiljöer för VRML, se SISUs rapport 96:15 ”Affärsapplikationer i 3D på Internet”.

4. Slutsatser

Hur lämpat är då VRML för interaktiva 3D-applikationer över WWW? Och hur väl fungerade den databaskoppling som fanns i applikationen? Vad är svagheterna med VRML? Är det möjligt att låta flera användare använda SISUs VRML-applikation samtidigt?

VRML är mycket väl lämpat för interaktiva 3D-applikationer. Det man intuitivt tänker på vad gäller VRML-applikationer och möjligheterna att köra dem på en PC över Internet är prestandan. Det finns två delar i prestandan som måste fungera väl bägge två. För det första själva beräk-ningarna. En PC med Pentium processor, vilket är att anse som en normal persondator idag, klarar att navigera i SISUs VRML-värld utan problem. Prestandan vad gäller överföringar skulle kunna bli bättre. Det som framförallt tar tid vid överföringar är texturerna som används för 3D-menyn. Så ett allmänt råd är att minimera användningen av texturer. Själva storleken på VRML-världen (utan texturer och VRML-effekterna) är ca 440 kb. Filen skulle kunna komprimeras till ca 50 kb (GZIP-komprimering hanteras automatiskt av VRML-läsare). Dessutom är ett binärt filformat med komprimering som är optimerat för just VRML på gång. Filformatet sägs kunna reducera storleken på VRML-filer ca 20-30 gånger. Dessutom hanterar VRML-läsarna dessa filer snabbare och effektivare än motsvarande VRML-filer i textformat. Detta skulle betyda att vi är nere i ca 20 kb för själva VRML-världen. En vanlig GIF-bild på t ex en logotyp är ofta betydligt större än detta och eftersom dessa används friskt kan storleken på VRML-filerna ej anses vara ett hinder. (Notera att texturer, ljud, m m fortfarande tar lika lång tid att överföra även om det kommande binära filformatet används.)

Databaskopplingen fungerade väl likaså. När SISUs applikation påbörjades fanns ej det externa Javagränssnittet. Detta innebar att det inte var helt trivialt att presentera en ”Spara inställningar” knapp för användaren då den måste byggas in i VRML-världen. Resultatet blev 3D-menyn. Mot slutet av projektets fas dök så det externa Javagränssnittet upp och alla sådana problem försvann. Med det externa Javagränssnittet är det möjligt att presentera t ex formulär och knappar som inte passar in i en 3D-värld. Avläsning av VRML-världens variabler från Javagränssnittet är mycket enkelt så Javakoden för att anropa ett CGI som sparar världskonfigurationen är enkel. Själva genererandet av världen var inte heller några större problem. Men om man i framtiden skulle vilja utnyttja GZIP-komprimering eller ett binärt filformat kan det vara svårare. Idén med GZIP-komprimering är att VRML-världen komprimeras *innan* användaren laddar ner den, d v s den totala tiden innan VRML-världen presenteras för användaren kortas ned eftersom överföringen av världen går fortare och tiden för uppackning är i det närmaste försumbar. Om världen istället GZIP-komprimeras *när den efterfrågas* skulle det innebära att användaren även måste vänta på att världen skall komprimeras innan den överförs. Detta innebär troligtvis längre total tid. För att gå runt detta skulle t ex databasvariabler kunna vara i separata filer som inkluderas med Inline-noden, men detta är troligtvis en mycket omständlig lösning. Problemet med binärt filformat utan komprimering blir ett annat. Då måste på något sätt notation för databasvariabler införas inuti den binära filen. Dessutom måste administratören av världen kunna gå in och ändra i variablerna. Alternativt att ett CGI läser in VRML-filen som en textfil och skriver den som en binärfil. Detta är inget stort problem och borde inte vålla någon systemutvecklare huvudbry.

I början av projektet ansågs den största svagheten med VRML vara just avsaknaden av möjlighet att presentera formulär och knappar. Detta är inget VRML-specifikt problem, utan

snarare något alla 3D-baserade miljöer lider av. Men VRMLs mycket väl lämpade scriptspråk tillsammans med det externa Javagränssnittet löste genast alla dessa problem då de dök upp. VRMLs svaga punkt kan nog istället sägas vara, eller bli, risken att Internetaktörerna inte följer standarden. Netscape är en mycket dominerande kraft i Internetvärlden och de har lovat att följa standarden men risken är att framförallt just Netscape kommer att presentera egna utbyggnader av standarden (framförallt i scriptspråket) liksom de gjort med HTML. *Om* det nu blir så att det kommer att finns en ”klicka här för den VRML-läsaren och här för den o s v” kan det medföra att företag drar sig för att använda VRML. Hur utvecklingen blir får framtiden utvisa.

Angående frågan om fleranvändarstöd för SISUs applikation finns inget riktigt entydigt svar. Det finns ett antal olika varianter av fleranvändarstöd för VRML 1.0 så rent intuitivt tycks fler-användarstöd inte vara något stort problem. Men skillnaden mellan VRML 1.0 och 2.0 är som bekant att i VRML 1.0 är alla objekt statiska medan det i VRML 2.0 finns externa Javagränssnitt, scripter, hög grad av interaktion, m m. Det går inte att bara låta de olika användarnas 3D-personer synas på varandras skärmar. Alla förändringar i världen måste även synas samtidigt på alla användares skärmar. Den enklare varianten av fleranvändarstöd som fanns till VRML 1.0 finns även till VRML 2.0. Denna enklare variant kan antagligen skapas på relativt kort tid av en någorlunda Javakunnig programmerare. Fullt utbyggt fleranvändarstöd där skaparen av VRML-världen själv kan avgöra helt fritt vilka förändringar i världen som skall vara gemensamma och vilka som skall vara lokala kommer att dröja ytterligare. Ett problem med VRML och fler-användarstöd är just standardiseringen. I och med att VRML 2.0 är standardiserad och ett fleranvändarstöd också kommer att bli det, eller åtminstone inkluderas som en bilaga till standardiseringen, måste i princip samtliga VRML-aktörer vara överens om lösningen. Och inom just området fleranvändarstöd finns det en myriad olika tillvägagångssätt och lösningar, alla med sina fördelar och nackdelar. Men i många andra applikationer där interaktionen med världen inte spelar en lika central roll som i SISUs VRML-applikation är det fullt möjligt för flera användare att använda världen gemensamt. Det existerar dessutom ett par produkter där fleranvändarstödet i VRML är längre gånget än den allra enklaste varianten, se SISU-rapport 96:15 ”Affärsapplikationer i 3D på Internet” för en kort genomgång av de olika varianter av fleranvändarstöd som existerar idag.