Java Accessibility

- tillgänglighet för funktionshindrade i ***Java***

1999-10-13, Anders Quist, Linné - Cell Network, Göteborg

Bearbetad av Mats Lundälv, DART 1999-2000

Vad menas med "Accessibility" / "tillgänglighet" ?

"Tillgänglighet"/"Accessibility" är en samlande beteckning för olika metoder, begrepp, program och hårdvaror som var för sig eller tillsammans bidrar till att IT-baserade system eller applikationer blir användbara oberoende av om man är funktionshindrad eller ej. Det finns ingen bestämd definition för exakt vad "Tillgänglighet"/"Accessibility" skall tillhandahålla gentemot användaren, bara en samling rekommendationer, API:er och praktiska exempel och tillämpningar.

Inom Java är det först och främst Sun som styr utvecklingen. Sun har, med stöd av bl.a. TRACE Research and Development Center och IBM, tagit fram ett API för Java som heter ”Accessibilty API”. Det syftar till att stödja och förenkla utvecklingen av tillgängliga tillämpningar genom att definiera ett gemensamt gränssnitt mellan en Java-applikation och diverse s.k. ”Assistive Technologies”. Exempel på sådana ”stödjande teknologier” kan vara alternativa styrfunktioner för gravt rörelsehindrade eller alternativ feedback för synskadade i form av bildskärmförstorare eller s.k. skärmläsare för syntetiskt tal eller brailledisplay.

IBM driver på utvecklingen inom området. Bland annat har man tagit fram riktlinjer för vad man bör tänka på när man bygger eller anpassar en befintlig applikation till att bli "tillgänglig". Man har också utvecklat, och utvecklar, en rad praktiska tillämpningar, t.ex. SVK, IBM's Java ”Self Voicing Kit” – verktyg för att förse Java-applikationer med ett auditivt gränssnitt, baserat på syntetiskt tal (mer information finns på http://www-3.ibm.com/able/svkalphapress.htm och http://www.alphaWorks.ibm.com/tech/svk). Tyvärr saknas f.n. stöd för svensk mjukvarutalsyntes (Infovox 230-330) baserat på Java Speech API.

Information om Java Accessiblity på nätet:

Lite länkar till olika platser på Internet där man kan få information som rör Accessiblity.

Senaste versionen av Accessibility API tillhandhåller Sun här:

http://java.sun.com/products/jfc/ och http://java.sun.com/products/jfc/accessibility.html

Allmänt om Accessible Software, div. tillämpningar m.m. hittar man hos IBM:
http://www-3.ibm.com/able/ och http://www-3.ibm.com/able/resources.htm

TRACE Centers utredningar och rekommendationer finns via:
http://www.tracecenter.org

Att tänka på när man utvecklar ”tillgängliga” applikationer i Java

När man börjar designa en Java-applikation som skall vara ”tillgänglig” skall man tänka på bl.a. följande:

- Använd i första hand **Swing**-baserade gränsnittskomponenter. (*Swing* är beteckningen på den senaste generationen av SUNs bibliotek med standardiserade gränsnittskomponenter för Java. Dessa ingår i JFC - Java Foundation Classes - inom ramen för det fritt tillgängliga JDK - Java Development Kit.) *Swing*-komponenterna implementerar vissa grundläggande delar av **Accessible**, ett interface som möjliggör att olika "Assistive Technologies" kan kommunicera med applikationen med hjälp av Accessibility-API:t. Man måste också tänka på att leva upp till vissa grundläggande krav, som att sätta ”Accessible name” och ”Accessible description” på sina komponenter.
- Om man gör en ”fler-fönster"-applikation så skall man vara medveten om att Java bara fångar events i det fönster som har fokus. Detta kan exempelvis vara ett problem om man har menyer i ett annat fönster än det som har fokus. Detta går att komma runt om man använder sig av AWTEventMonitor alternativt SwingEventMonitor i Accessiblity-API:t. Man bör ta i beaktande att AWTEventMonitorn tar emot och fångar samtliga events i en applikation. Man får därför tänka på att filtrera events utifrån vilken komponent eller vilket fönster som har fokus.
- Samtliga funktioner som finns i applikationen skall gå att utföra ”Mouseless”, dvs med enbart tangentbordet. Vissa befintliga äldre gränssnittskomponenter i Java är rent musstyrda. Man bör ha detta i åtanke om man exempelvis överväger att använda sig av AWT-komponenter (AWT = Abstract Windows Toolkit). Använd hellre en motsvarande Swing-komponent!
- Applikationen får inte störa befintliga grundanpassningar i operativsystemet, som exempelvis "Tröga tangenter" ("Sticky Keys") m.m. (i Windows- och Mac-miljöerna).

Riktlinjer

Nedan följer några riktlinjer till hjälp då man skall göra en ”tillgänglig” applikation.

1.0 Tangentbordsaccess

1.1 Det skall finnas tangentbordsmotsvarigheter till samtliga musstyrda operationer.
1.2 Gränssnittet bör vara väl genomtänkt med möjlighet att nå och styra samtliga gränssnittskomponenter via tangentbordet på ett strukturerat och effektivt sätt (Tab-ordning mm).
1.3 Snabbtangenter skall vara definierade för funktioner som används ofta.
1.4 Dokumentera all tangentbordskommandon.

2.0 Genomskinlighet gentemot Accessibility-anpassningar i OS

2.1 Programmet får ej störa eller påverka befintliga funktioner för anpassad tillgänglighet i operativsystemet (Tröga tangenter/Sticky Keys etc).

3.0 Objektinformation

3.1 Det objekt som har fokus skall vara tydligt markerat, såväl för användaren som programmatiskt för eventuell kompletterande anpassningar (Assistive Technology).
3.2 Enkla och tydliga texter om hur man kan påverka eller använda ett objekt.

4.0 Ikoner

4.1 Man bör alltid ha text i samband med ikoner - antingen som skrivbordsikoner (som exempelvis skrivbordet i Windows95) eller som ”tool-tips” eller ”bubbel”-hjälp.
4.2 Ikoner som illustrerar en viss betydelse bör användas konsistent och konsekvent.

5.0 Layout

5.1 Textetiketter (”Text labels”) bör placeras nära det objekt som det skall associeras med.
5.2 Objekt som hör ihop bör grupperas på ett tydligt, logiskt/praktiskt och lättnavigerat sätt.

6.0 Ljud

6.1 Alla varningsljud etc. bör ha en visuell motsvarighet (ex.: varningslampa, skärmen blinkar,…).
6.2 Om information presenteras auditivt så bör information även kunna fås i textform. Samma gäller även för eventuella videosekvenser.
6.3 Användaren ska själv kunna styra volymen och stänga av ljudet.

7.0 Skärmvisning

7.1 Använd inte färg som enda informationsbärare eller för att ge feedback.
7.2 Ge helst användaren möjlighet att själv kunna ange färg, typsnitt mm för text eller gränssnittskomponenter.

8.0 Timing

8.1 Låt användaren själv bestämma blinkfrekvenser för exempelvis textinsättningsmarkörer..
8.2 Man bör inte ha moment i applikationen där responsen till programmet måste ges inom ett visst begränsat tidsintervall. Vänta istället på användarens respons eller förläng tiden.

9.0 Dokumentation

9.1 Ge klar och exakt information om samtliga tillgänglighetsstöd i applikationen.
9.2 Dokumentationen för applikationen bör finnas i antingen ren ASCII-text eller som HTML.

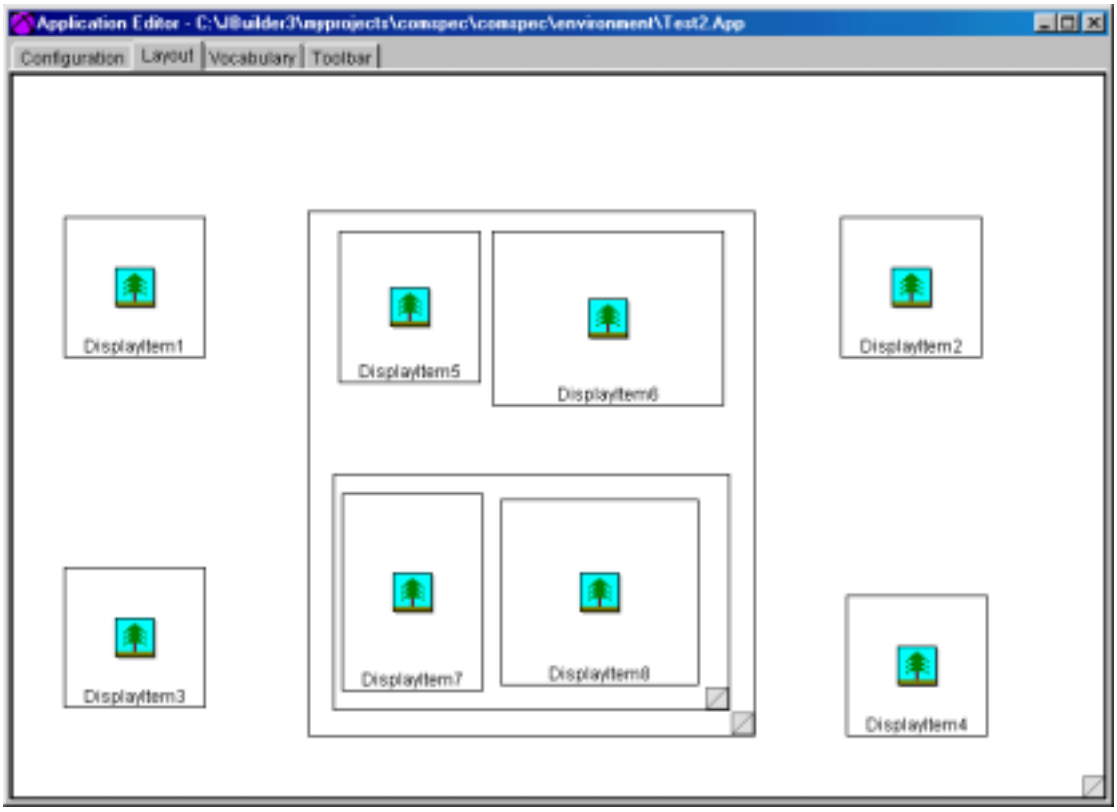
10.0 Verifiera

10.1 Kontrollera att alla funktioner går att utföra enbart via tangentbordet på ett tillfredställande vis.
10.2 Testa programmet med diverse anpassningsteknik (skärmförstoring, tröga tangenter etc.)

Accessibility i ComLink

Målet

För att få praktiska erfarenheter av tillgänglighetsproblematiken valde vi att göra en anpassning av gränssnittet i ComLinks redigeringsdel. Målet blev att göra det möjligt för en användare att enbart via tangentbordet kunna navigera sig igenom komponenterna i subsystemet ”Layout Editor". Subsystemet är i första hand en grafisk editor som låter användaren bygga eller anpassa utseendet hos en applikation i ComLink.



Layout-editorn i ComLink. Exempel på CardStack i en CardStack.

Beskrivning av subsystemet

I Layout-editorn byggs en applikations gränssnitt baserat på en sidstruktur. Varje applikation innehåller alltså en eller flera sidor. Sidan utformas sedan med hjälp av en rad grundkomponenter (LayoutItems). En av dessa är CardStackEditor som låter användaren påverka en CardStack-klass. CardStack (korthögen) är en s.k. "container"-komponent som kan innehålla ett eller flera Cards (kort), och där varje kort i sin tur kan innehålla en godtycklig mängd andra komponenter. Dessa kan exempelvis vara DisplayItems, EditorAreas eller ytterligare CardStacks för att bara nämna några. Denna uppbyggnad gör att dom olika komponenterna kan ha ett hierarkiskt förhållande till varandra, vilket ställde ytterligare krav på lösningen i form av navigation över flera nivåer.

Lösning
Genom att lägga till ett antal metoder i CardStackEditor klassen för att programmatiskt kunna styra vilken komponent som skall markeras fångas sedan en specifik tangent och respektive metod anropas.

Metoderna som lades till i CardStackEditor var:

public void selectNext()
Markera nästa komponent i ordningen efter den redan markerade. Om ingen komponent är markerad så markeras den första. Ordningen på komponenterna motsvaras av den ordning dom skapats.

public void selectPrevious()
Markera komponenten innan den redan markerade. Om ingen komponent är markerad så markeras den första.

public void selectFirst()
Markerar den första komponenten i ordningen. Anropas inte direkt.

public void select(java.awt.Component)
För att markera en specifik komponent. Anropas inte direkt.

public void goIn()
Om den markerade komponenten är en CardStack så görs denna aktiv och första komponenten i första Cardet markeras.

public void goOut()
Om det finns en giltig CardStack förälder till den aktiva så görs den aktiv och den före detta aktiva blir den markerade.

Accessibility - Exempel

Dessa exempel fungerar enbart med JDK/JRE1.2 och Java Accessibilty API 1.1 eller senare.

Exempel 1 – Accessible name
Hur man sätter ”accessible name” på en ikon-knapp:

```
JButton button = new JButton(new ImageIcon("image.gif"));
button.setToolTipText("Button Name");


// Om tool tip används till något annat kan man specifikt sätta,
// accessible namnet på komponenten på följande sätt.
button.getAccessibleContext().setAccessibleName("Button Name");
```

Exempel 2 - AWTEventMonitor

Här följer ett exempel för hur man kan använda sig av AWTEventMonitor för att fånga KeyEvents i en JFrame.

Applikationen registrerar sig som KeyListener hos AWTEventMonitor och skriver ut till en JTextArea när den får ett KeyEvent.

```
import java.awt.event.*;
import com.sun.java.accessibility.util.*;
import javax.swing.*;


public class AWTEventFrame extends JFrame implements
  KeyListener,
  GUIInitializedListener {


  JTextArea textArea = new JTextArea();
  JScrollPane scrollPane;


  public AWTEventFrame() {
    this.setTitle("AWTEventFrame");
    this.setSize(300,400);

    scrollPane = new JScrollPane(textArea);
    this.getContentPane().add(scrollPane);
```

```
    // Går bara att lägga till Listeners i AWTEventMonitor när gränsnittet
    // är färdiginitialiserat
    if (EventQueueMonitor.isGUIInitialized()) {

      // Gränsnittet är klart
      AWTEventMonitor.addKeyListener(this);
    }
    else {

      // Nej, vi väntar på att det blir klart
      EventQueueMonitor.addGUIInitializedListener(this);
    }
  }


 public static void main(String[] args) {
    AWTEventFrame frame = new AWTEventFrame();
    frame.setVisible(true);
  }



  // Implementationen av KeyListener
  public void keyTyped(KeyEvent e) {
    textArea.append("Key typed \n");
    e.consume();
  }

  public void keyPressed(KeyEvent e) {
    textArea.append("Key pressed: " + e.getKeyText(e.getKeyCode()) + "\n");
    e.consume();
  }

  public void keyReleased(KeyEvent e) {
    textArea.append("Key released: " + e.getKeyText(e.getKeyCode()) + "\n");
    e.consume();
  }



  // Implementationen av GUIInitializedListener
  public void guiInitialized() {
    EventQueueMonitor.removeGUIInitializedListener(this);
    AWTEventMonitor.addKeyListener(this);
  }
}
```