# A model-engineering approach to implementing Personal Universal Controller

Yuan Jin

April 30, 2008

# Outline

- Introduction
- Problems analysis — Jeffrey Nichols' work
- From androiddevice to androidgui
- From unconstrained to constrained
- From model to files

My work

- Future work
- References

# The issue

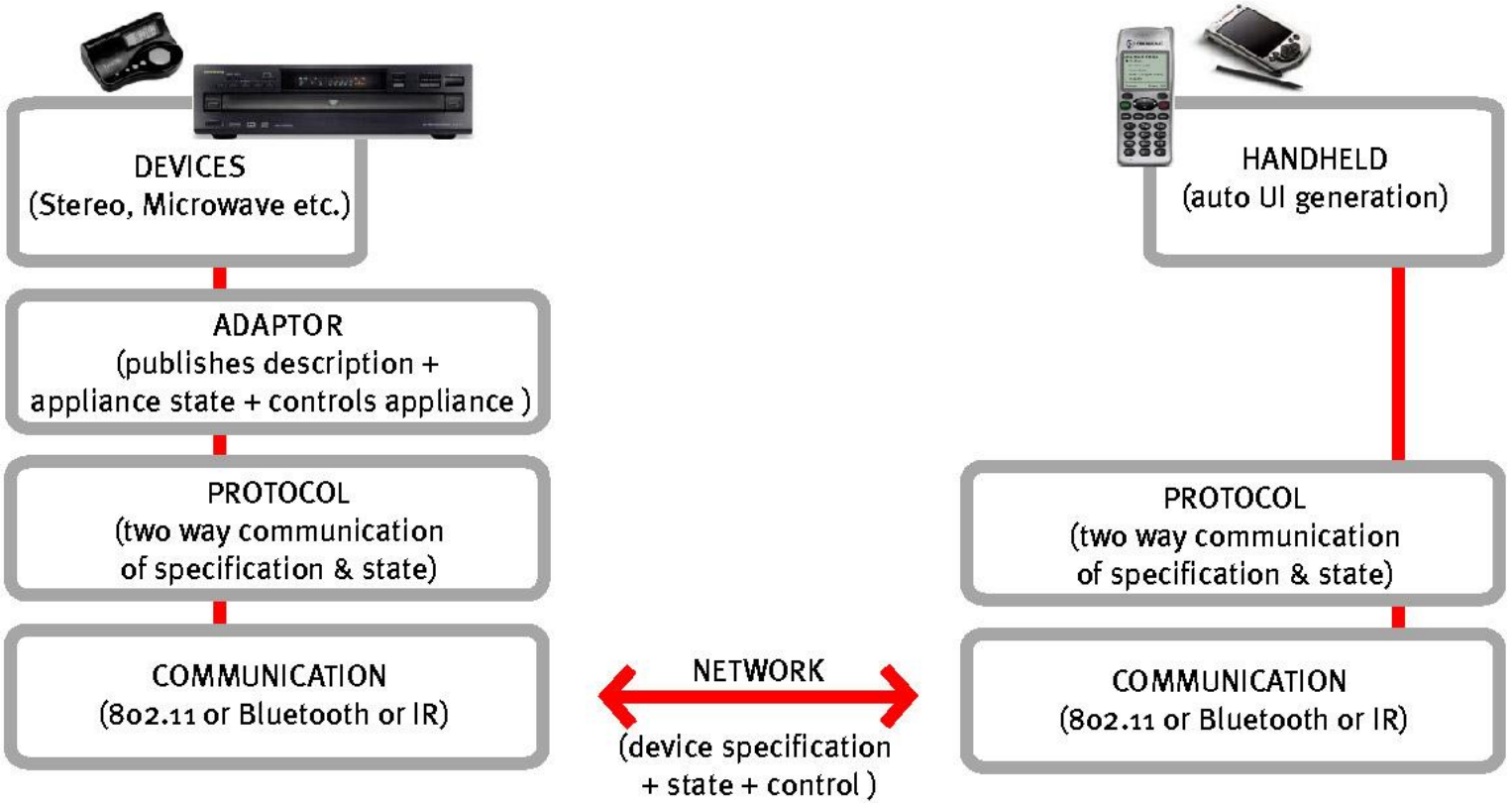
The world is congested with complex and  idiosyncratic interfaces

# PUC and its component

It is a two-way communication

Devices describe their functions

**PUC creates interface based on description and controls it**



# Application specification and communication protocol

**Specification Language**

```
<?xml version="1.0" encoding="UTF-8"?>
<spec xmlns="puc.xsd"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    name="Audiophase 5 CD Stereo">

 <groupings>

  <state name="PowerState">
   <type name="OnOffType">
    <valueSpace>
     <boolean/>
    </valueSpace>
    <valueLabels>
     <map value="false">
      <label>Off</label>
     </map>
     <map value="true">
      <label>On</label>
                        </map>
                       </valueLabels>
   </type>
   <labels>
    <label>Stereo Power</label>
    <label>Power</label>
                       <label>Powr</label>
                       <label>Pwr</label>
   </labels>
   <priority>10</priority>
  </state>
```

- Is abstract: makes no reference to interface particulars
- Is concise: only required type information is transmitted
- Parsing XML

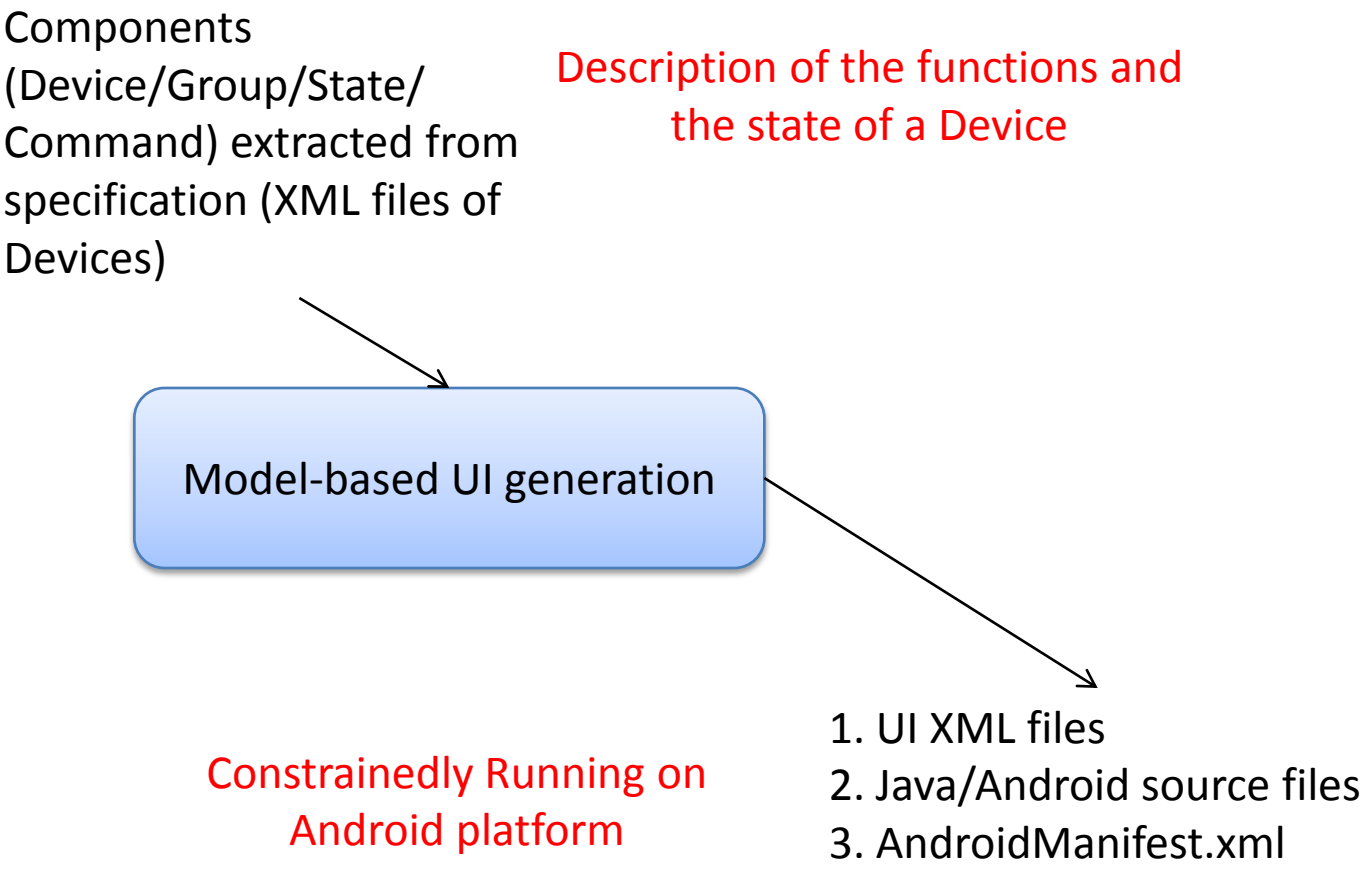
**Communication Protocol**

```
<message>
  <state-change-request>
    <state>(state)</state>
    <value>(value)</value>
  </state-change-request>
</message>

<message>
  <state-change-notification>
    <state>(state)</state>
    <value>(value)</value>
  </state-change-notification>
</message>

<message>
  <spec-request/>
</message>

<message>
  <device-spec>
    <spec>(spec)</spec>
  </device-spec>
</message>

<message>
  <full-state-request/>
</message>
```

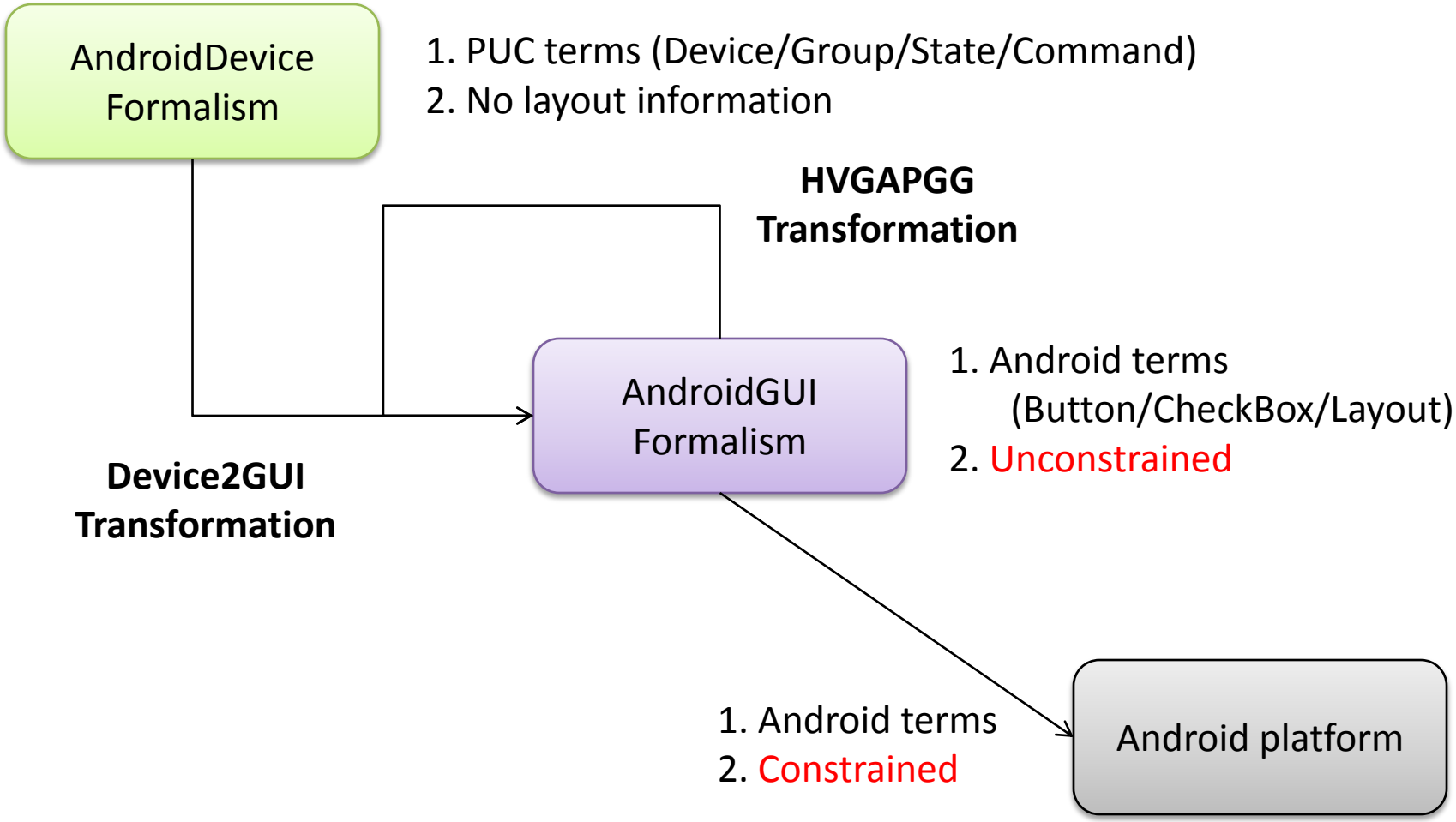
# Outline

- Introduction
- Problems analysis
- From androiddevice to androidgui
- From unconstrained to constrained
- From model to files
- Future work
- References

# Overview


Components (Device/Group/State/ Command) extracted from specification (XML files of Devices)
Description of the functions and the state of a Device
Model-based UI generation
Constrainedly Running on Android platform
1. UI XML files
2. Java/Android source files
3. AndroidManifest.xml

# Model-based UI generation



# Unconstrained and constrained layouts


**Unconstrained**


**Constrained**

# Problem forming

- Creation of a PUC formalism
- Creation of an Android formalism
- Transformation between above two formalisms
- Transformation from unconstrained to constrained layout
- Output from model to files

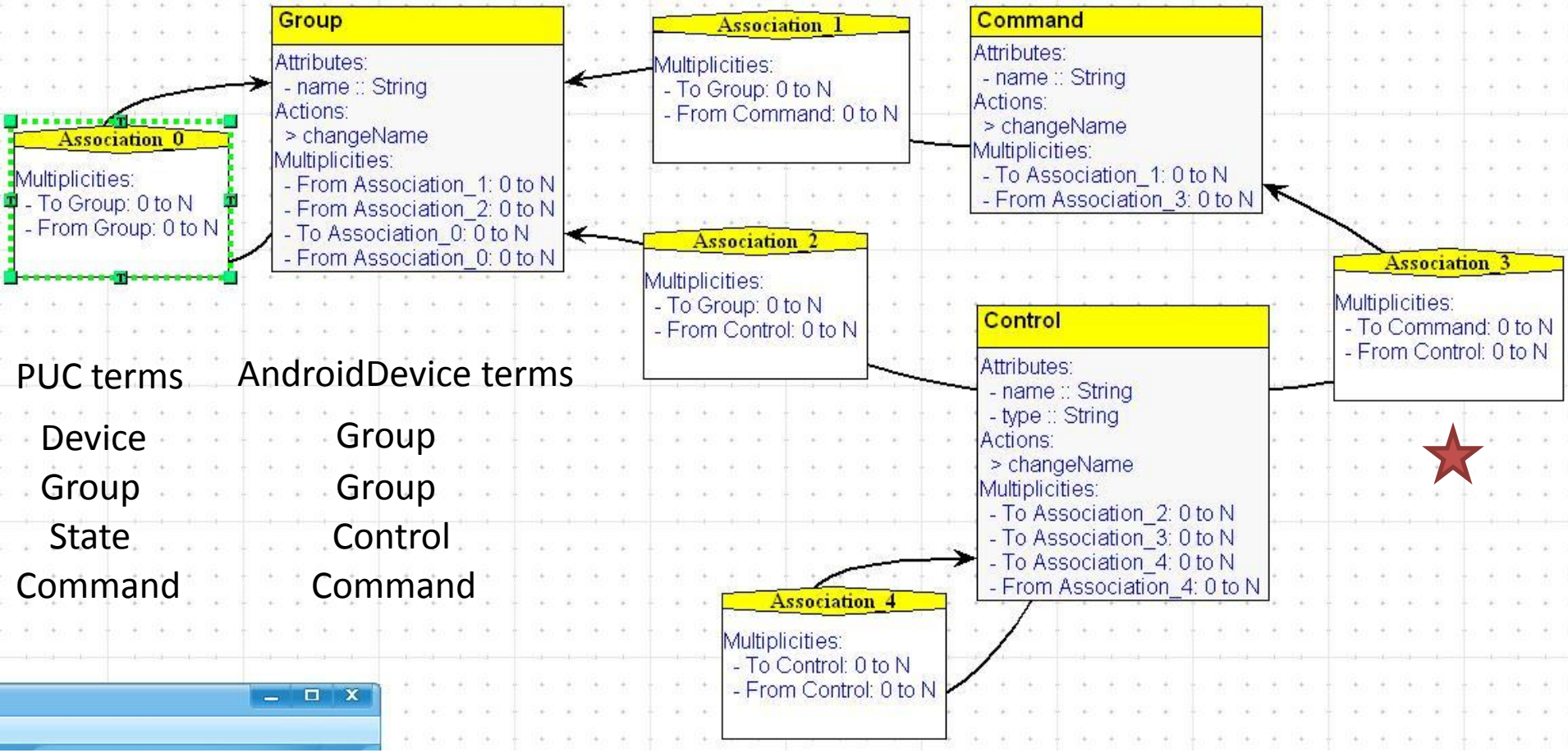
Benefit:

1. Easy to program
2. Better for practical use
3. Available for other platforms

# Outline

- Introduction
- Problems analysis
- From AndroidDevice to AndroidGUI
- From unconstrained to constrained
- From model to files
- Future work
- References

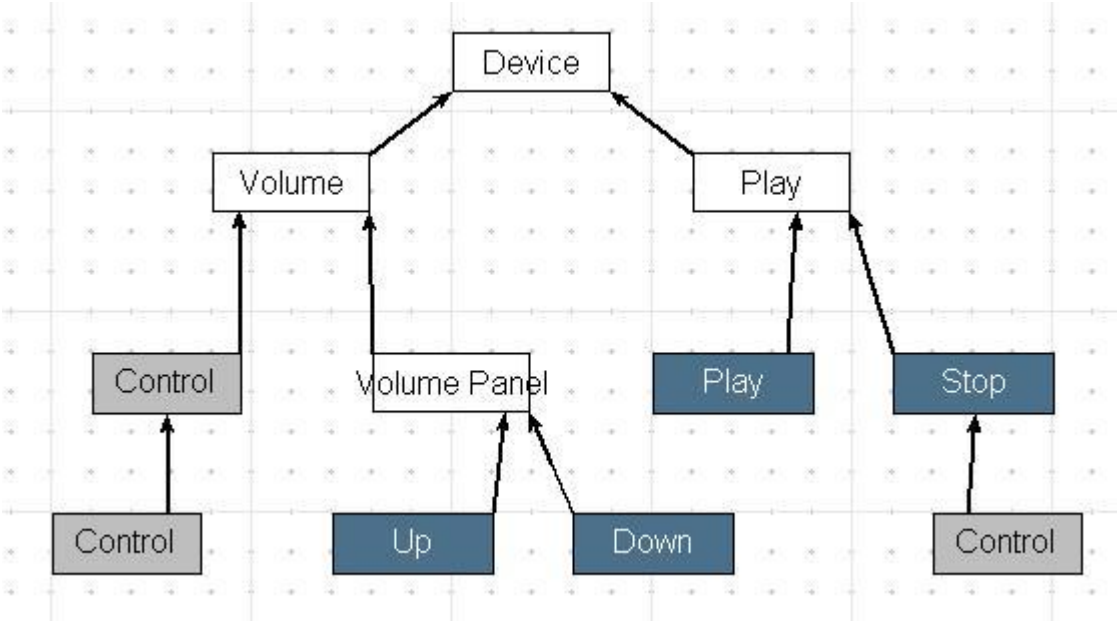
# AndroidDevice formalism



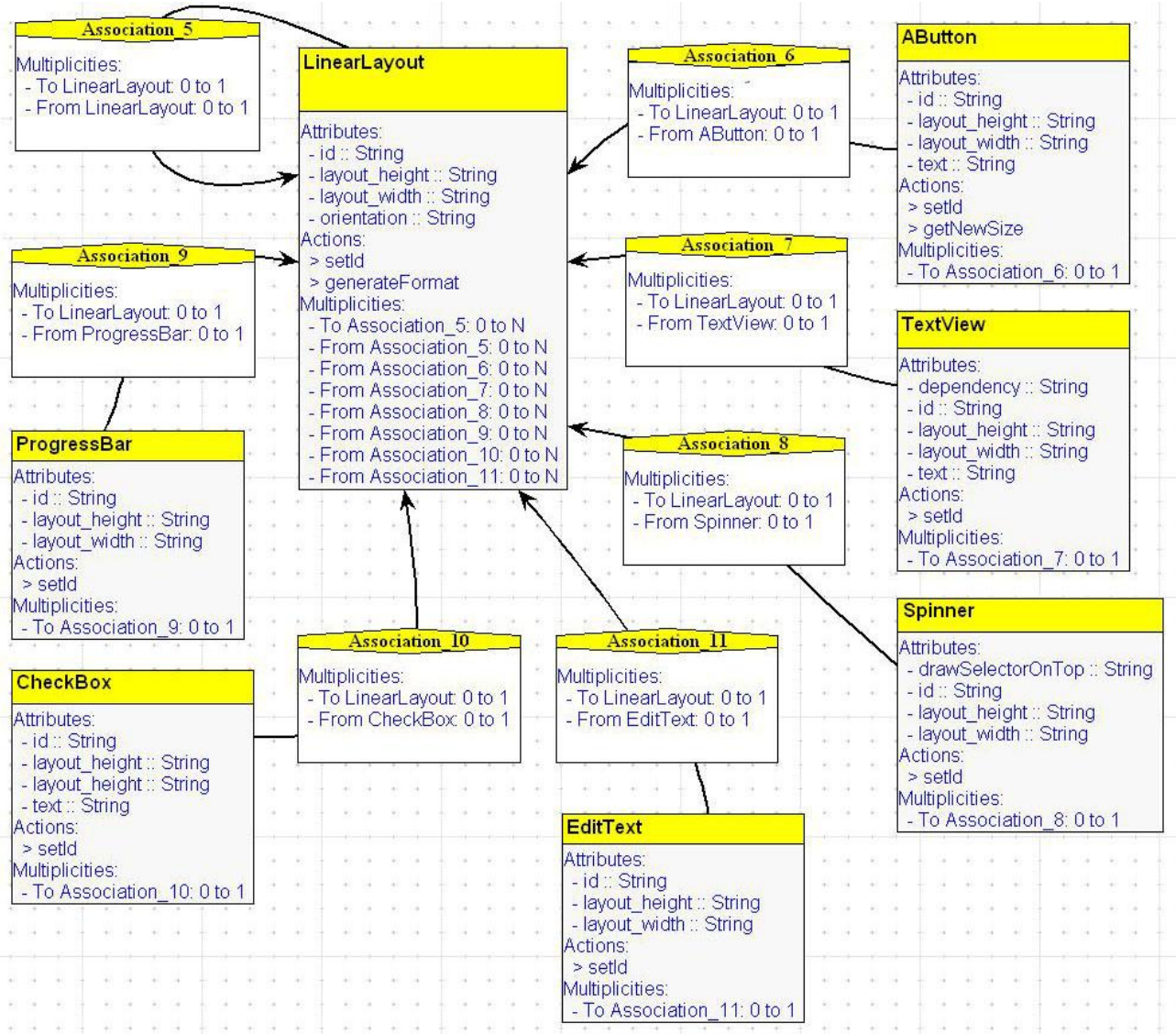

| PUC terms | AndroidDevice terms |
|---|---|
| Device | Group |
| Group | Group |
| State | Control |
| Command | Command |

# An example in AndroidDevice

Purpose of this formalism:
To use components included in the PUC XML specification file to describe the functions and state of a Device.

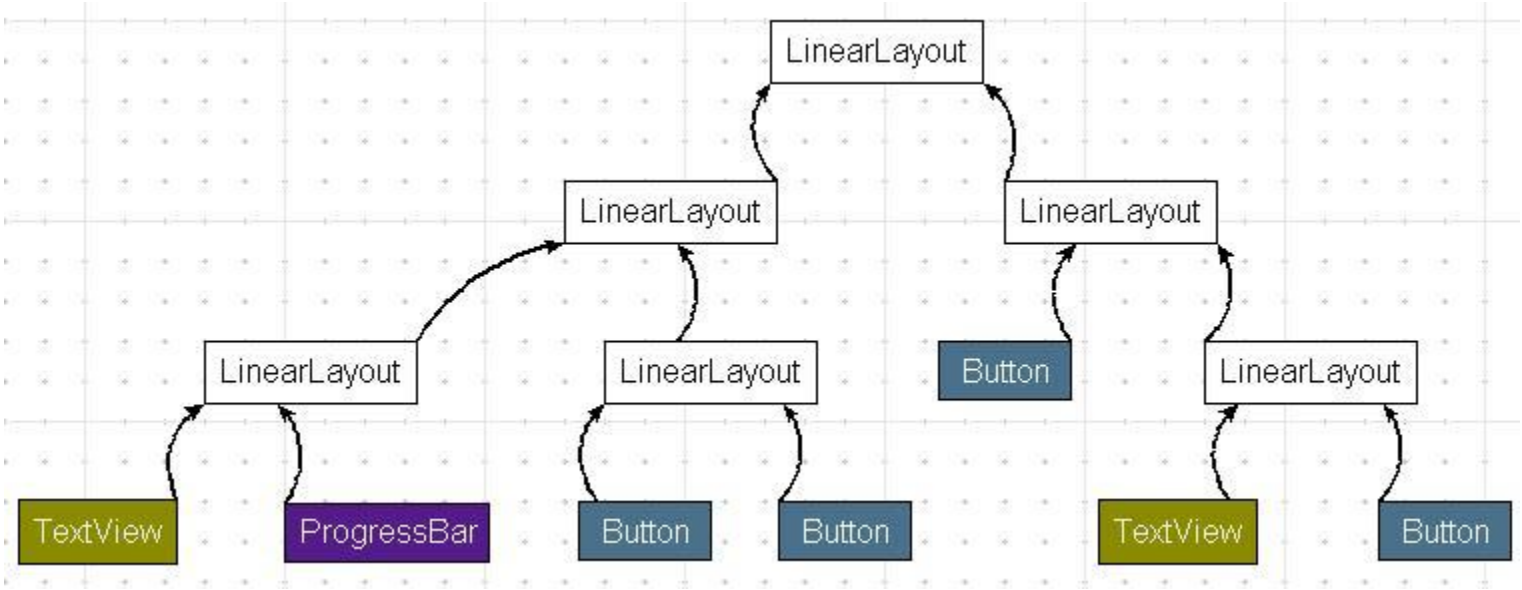


# AndroidGUI formalism



# An example in AndroidGUI

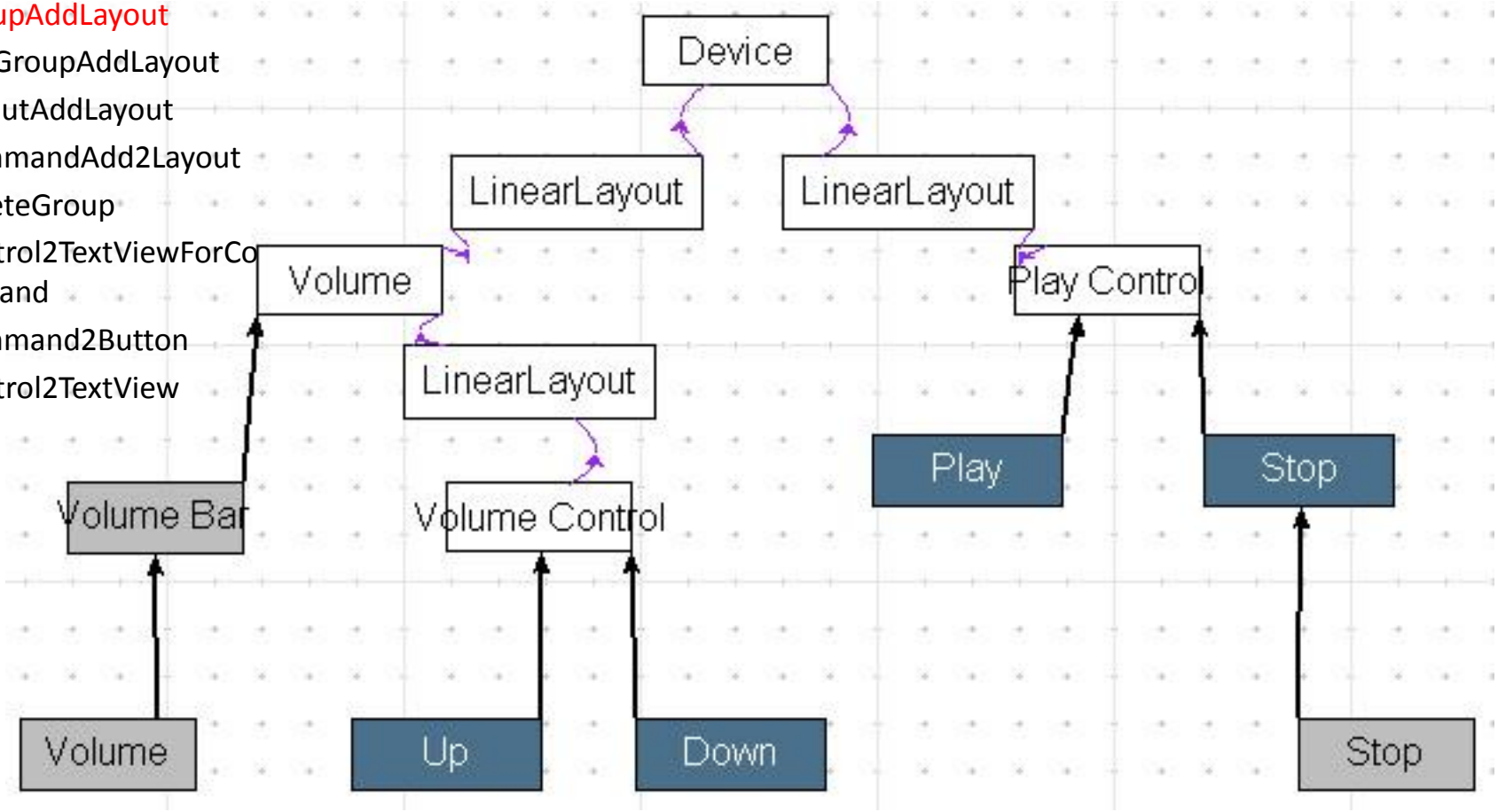
Purpose of this formalism: To model the user interface layout using specified components that are similar to Android's




This formalism can work independently to draw the Android UI graphically.

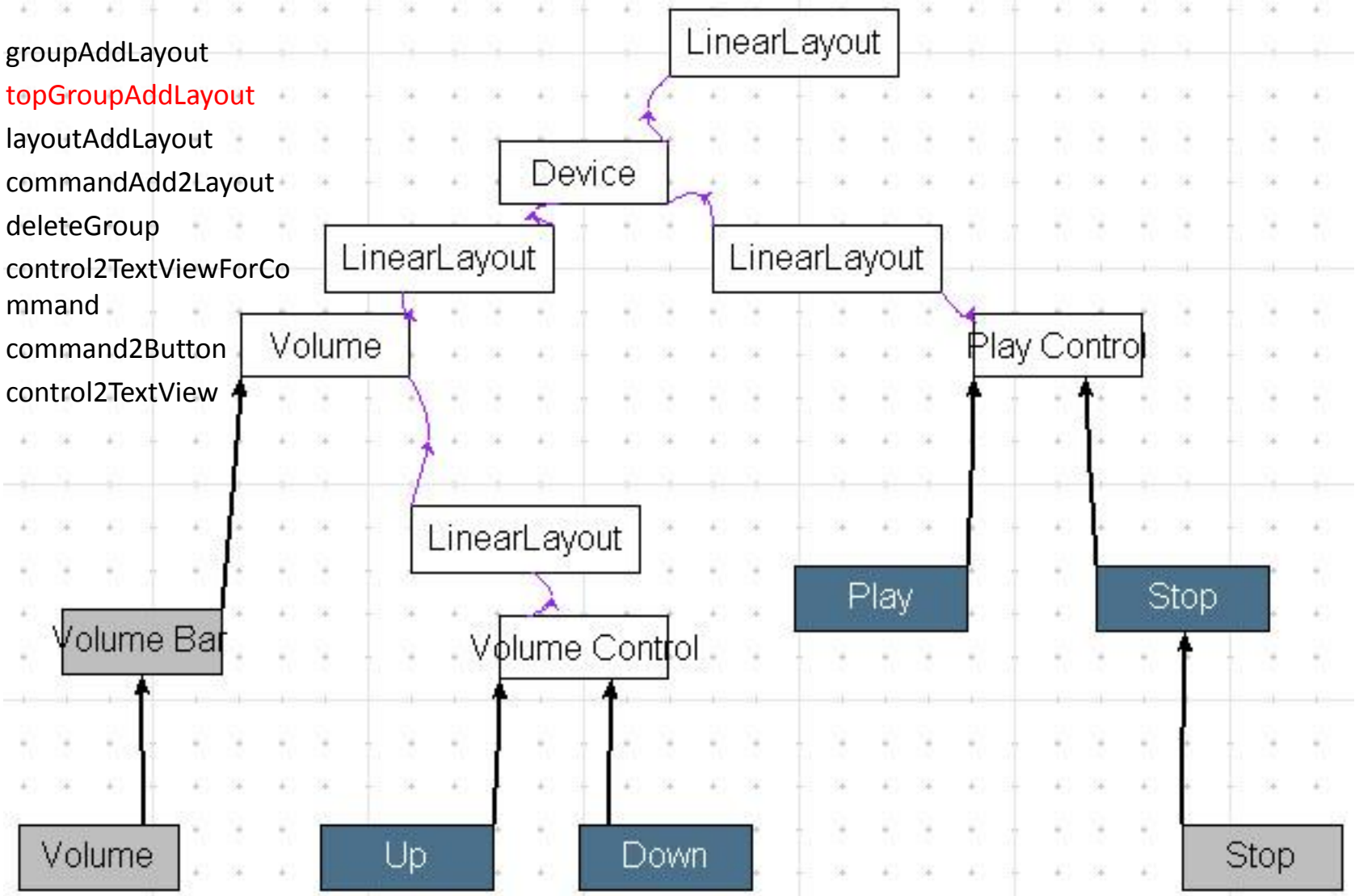
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



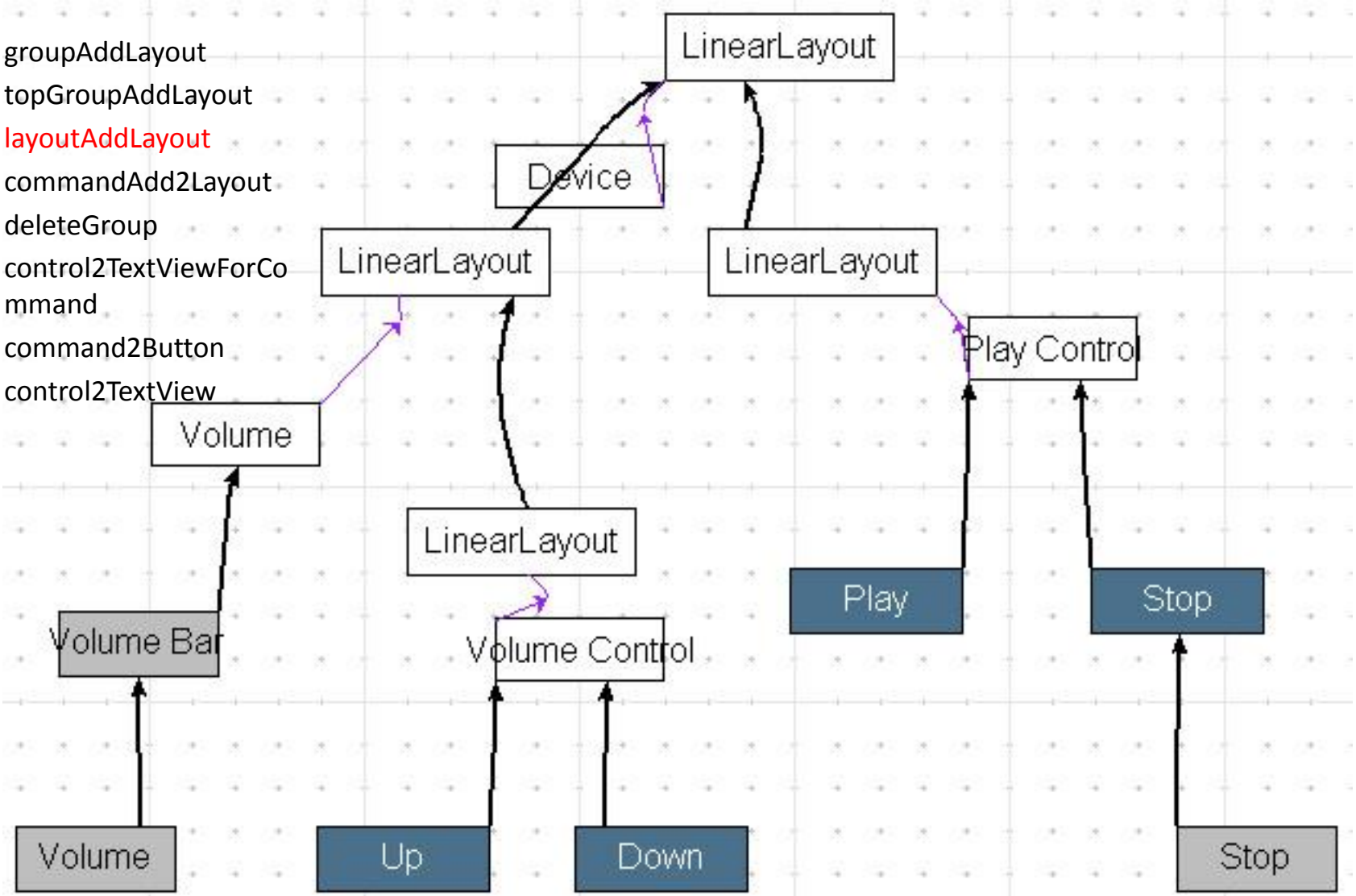
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



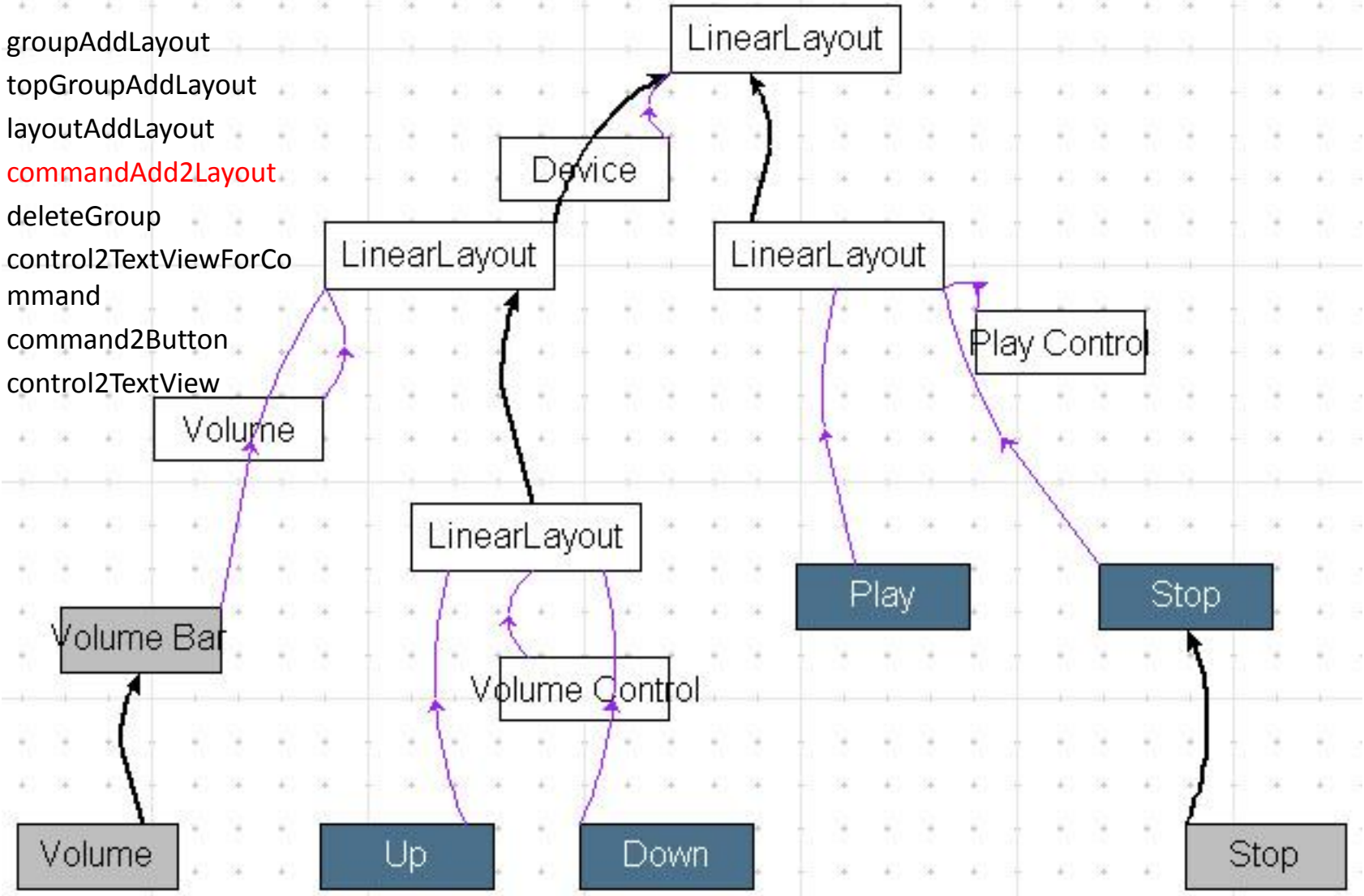
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



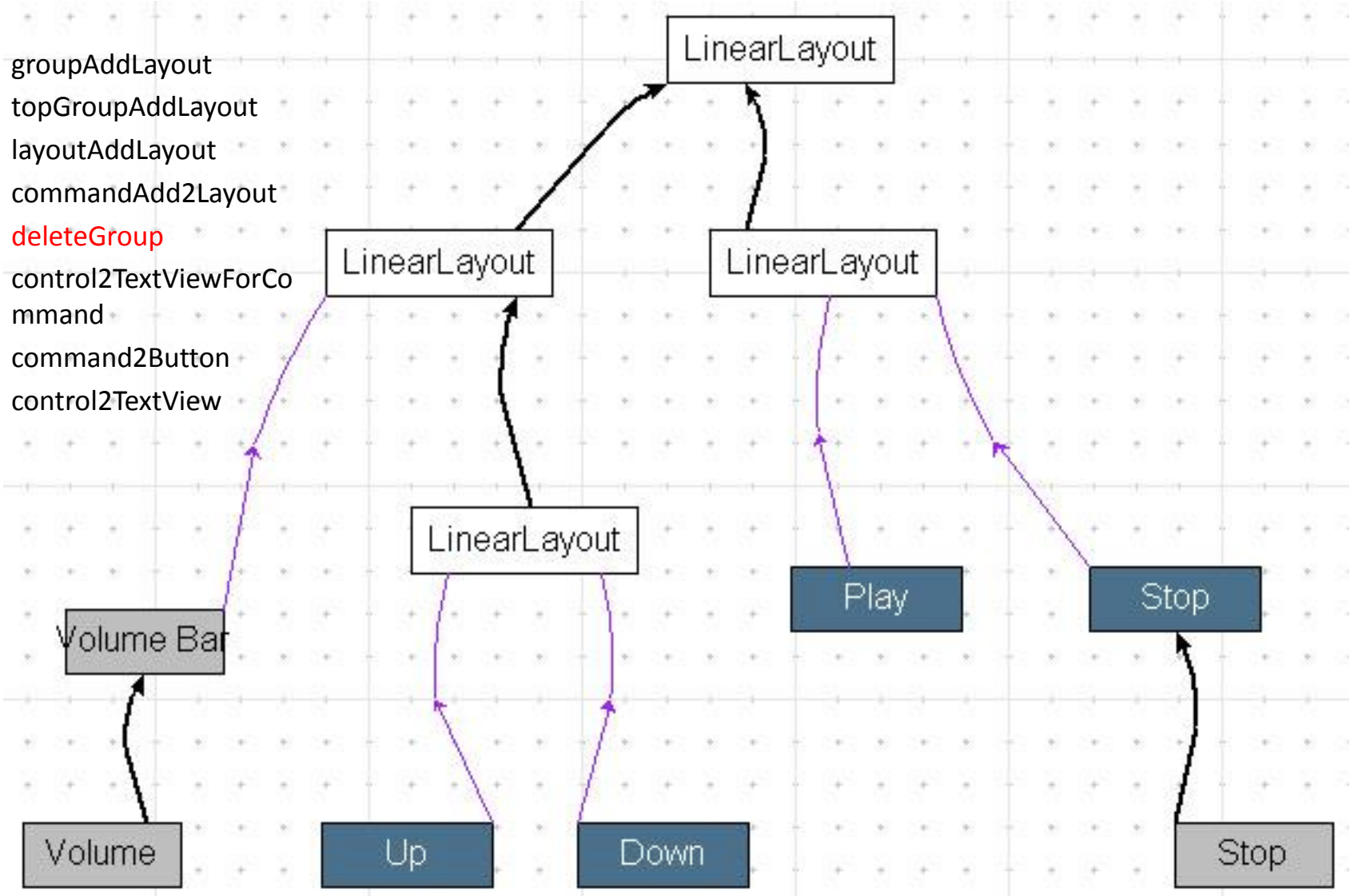
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



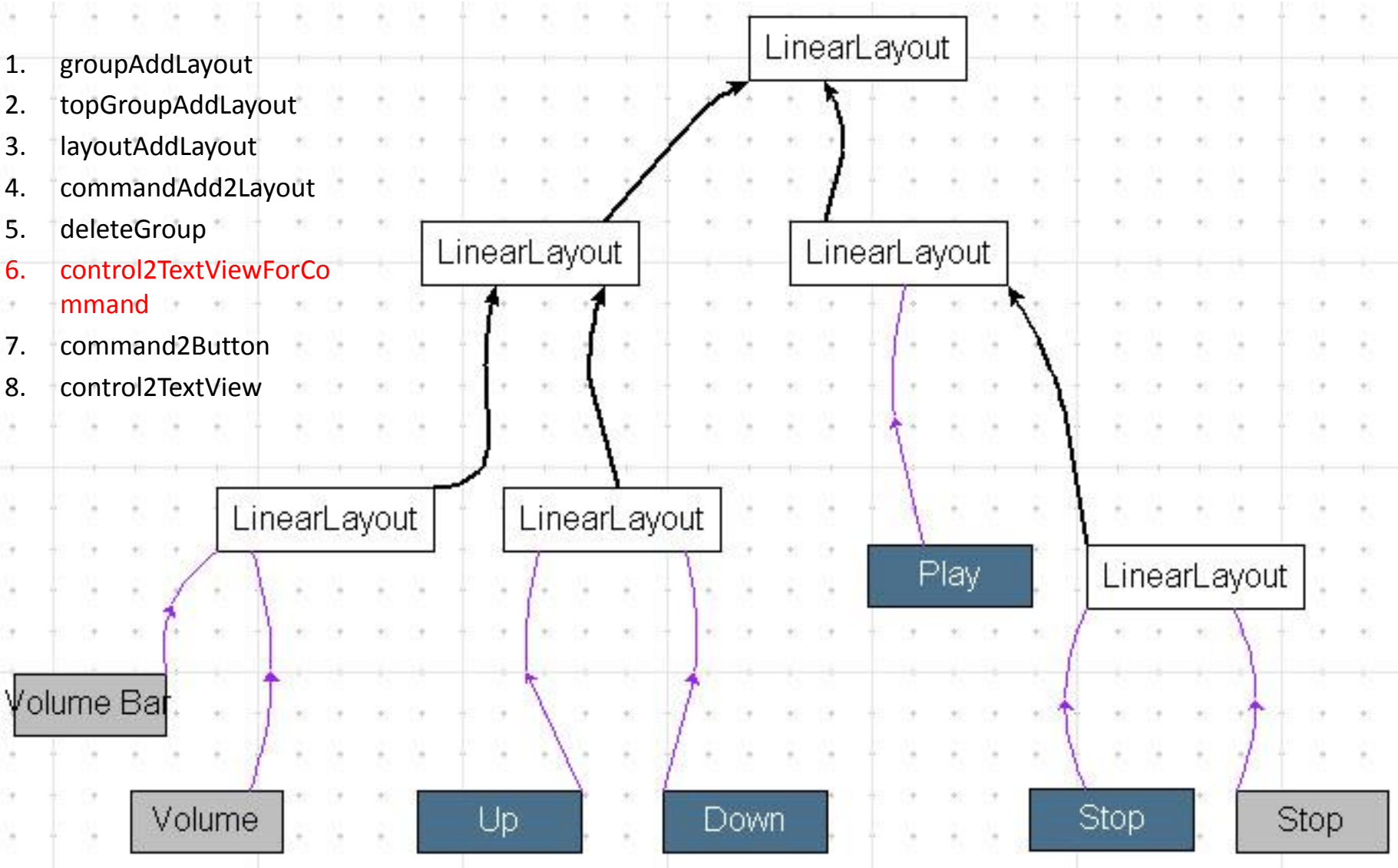
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



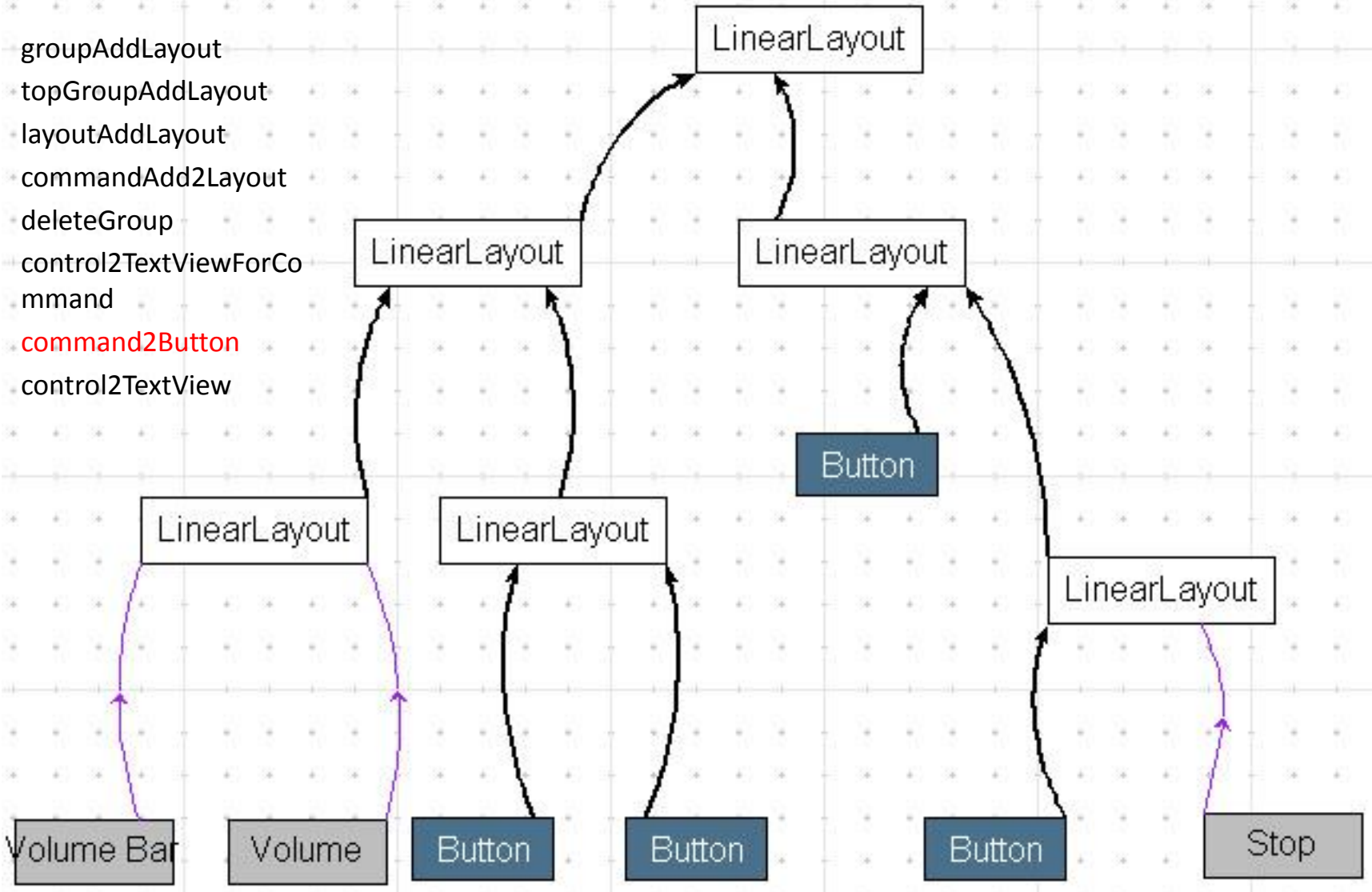
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



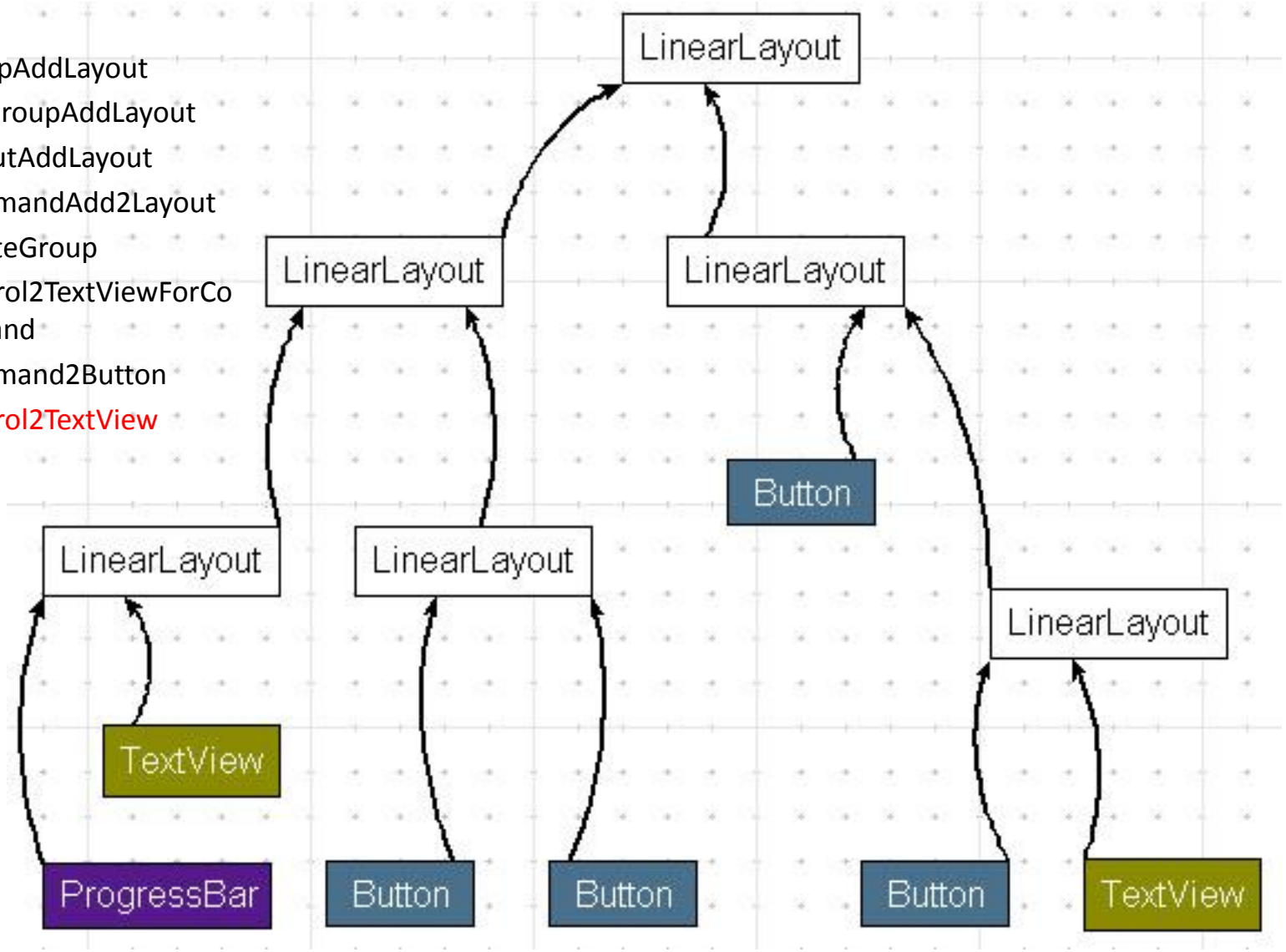
# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



# Device2GUI Transformation

1. groupAddLayout
2. topGroupAddLayout
3. layoutAddLayout
4. commandAdd2Layout
5. deleteGroup
6. control2TextViewForCommand
7. command2Button
8. control2TextView



# Outline

- Introduction
- Problems analysis
- From AndroidDevice to AndroidGUI
- From unconstrained to constrained
- From model to files
- Future work
- References

# Unconstrained and constrained layouts


**Unconstrained**

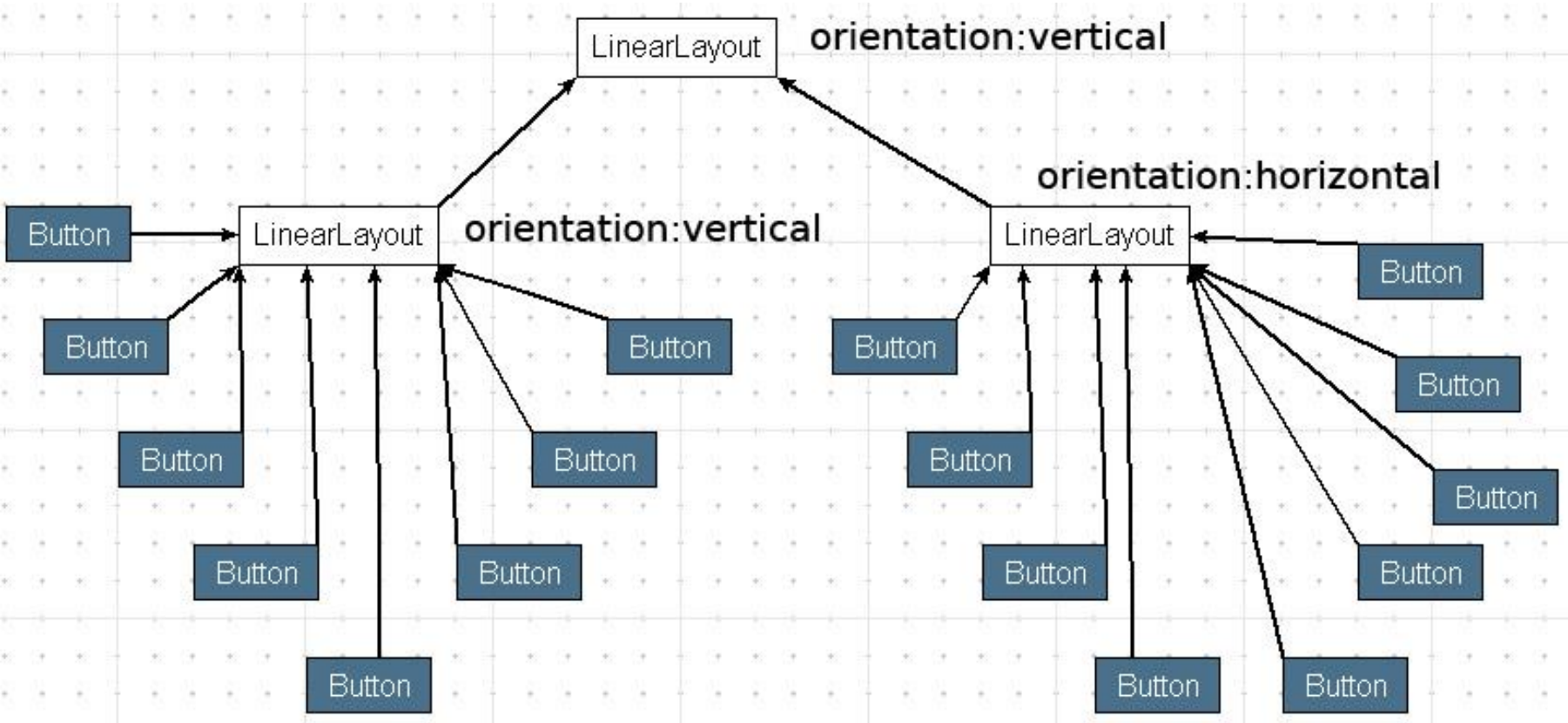

**Constrained**

# Layout Transformation

- If a small structure, this model can be used directly
- If a large one, needs some transformation regarding the screen size/layout information
- Horizontal and vertical transformations
- More difficult than the previous transformation
- Demo specified HVGA-Portrait (HVGAPGG)

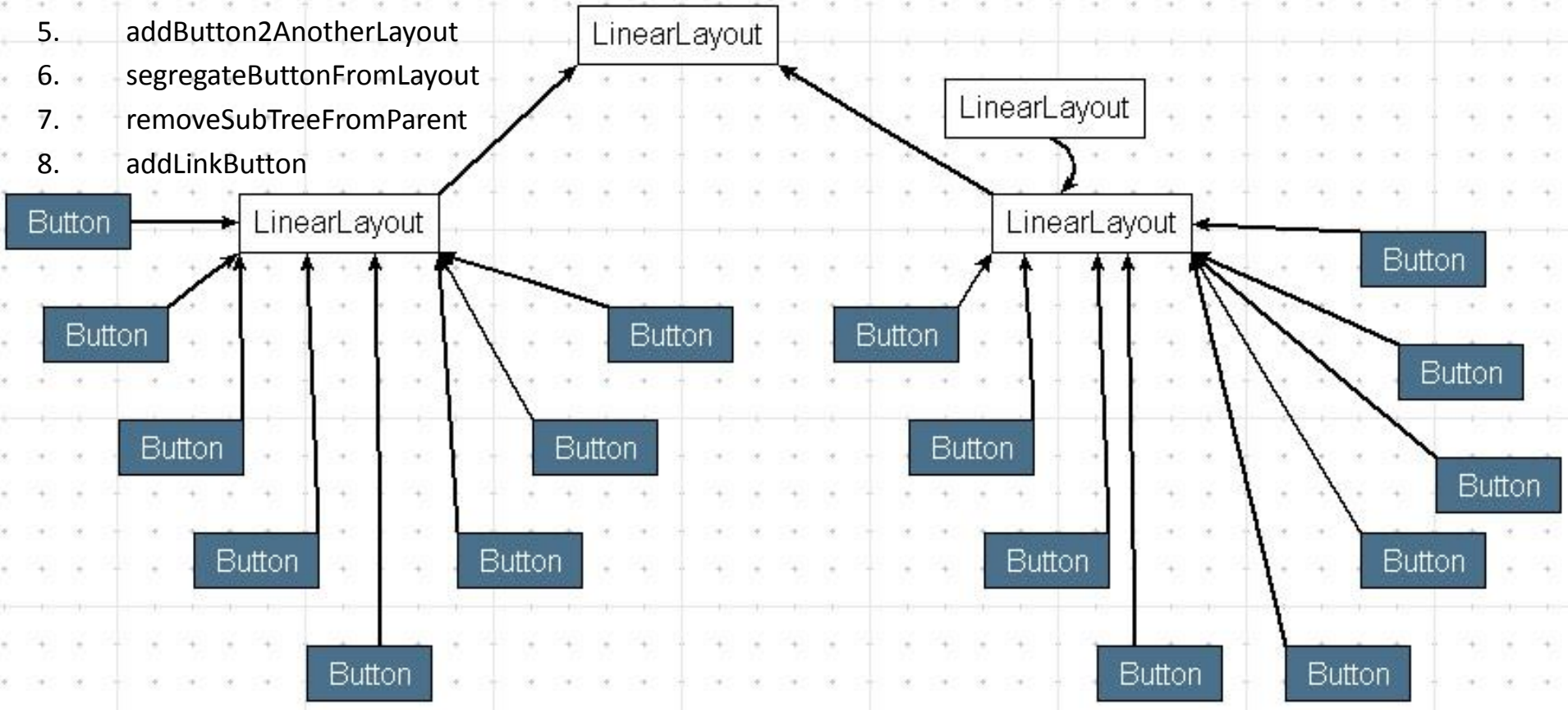
# An example of HVGAPGG Transformation

HVGA-Portrait allows:
Horizontal: 4 components  Vertical: 8 components



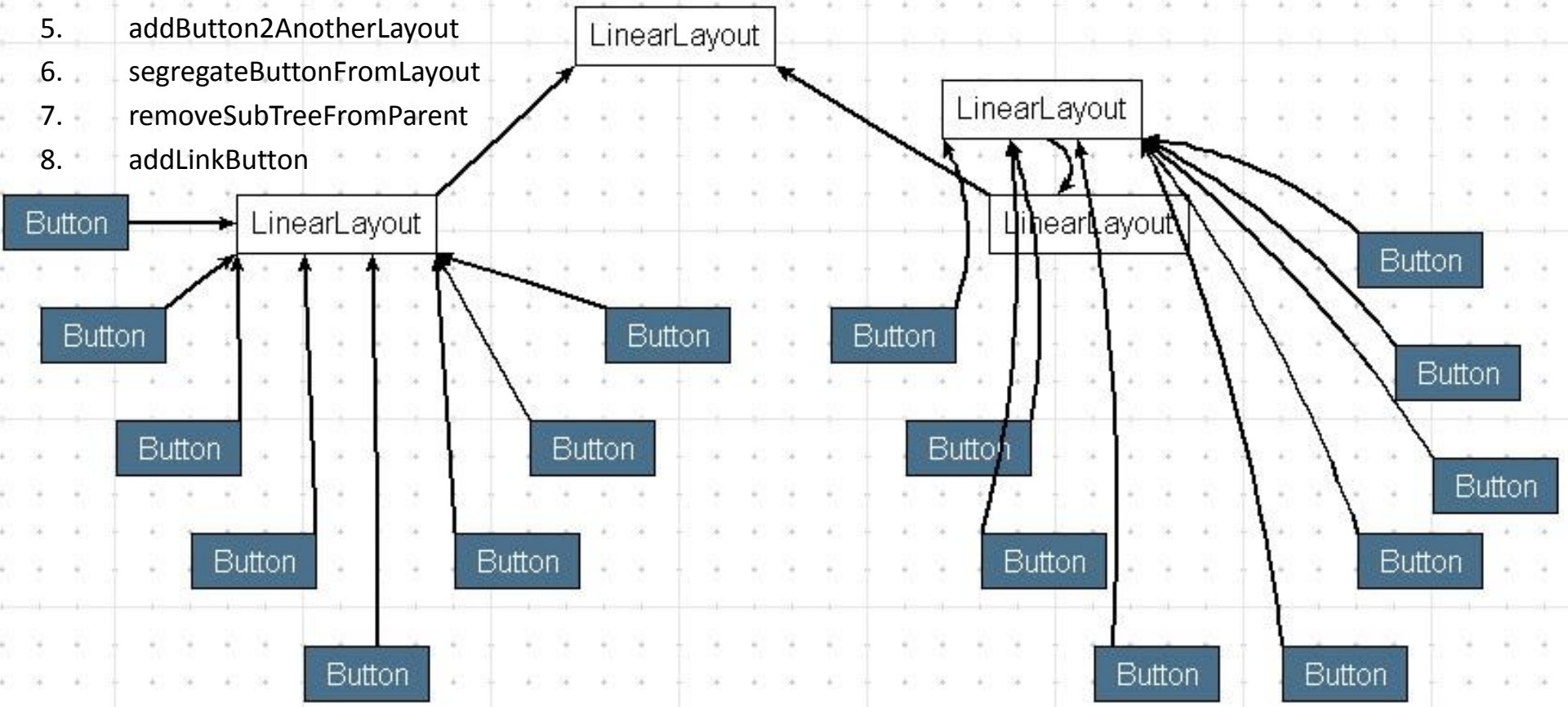
# Horizontal Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



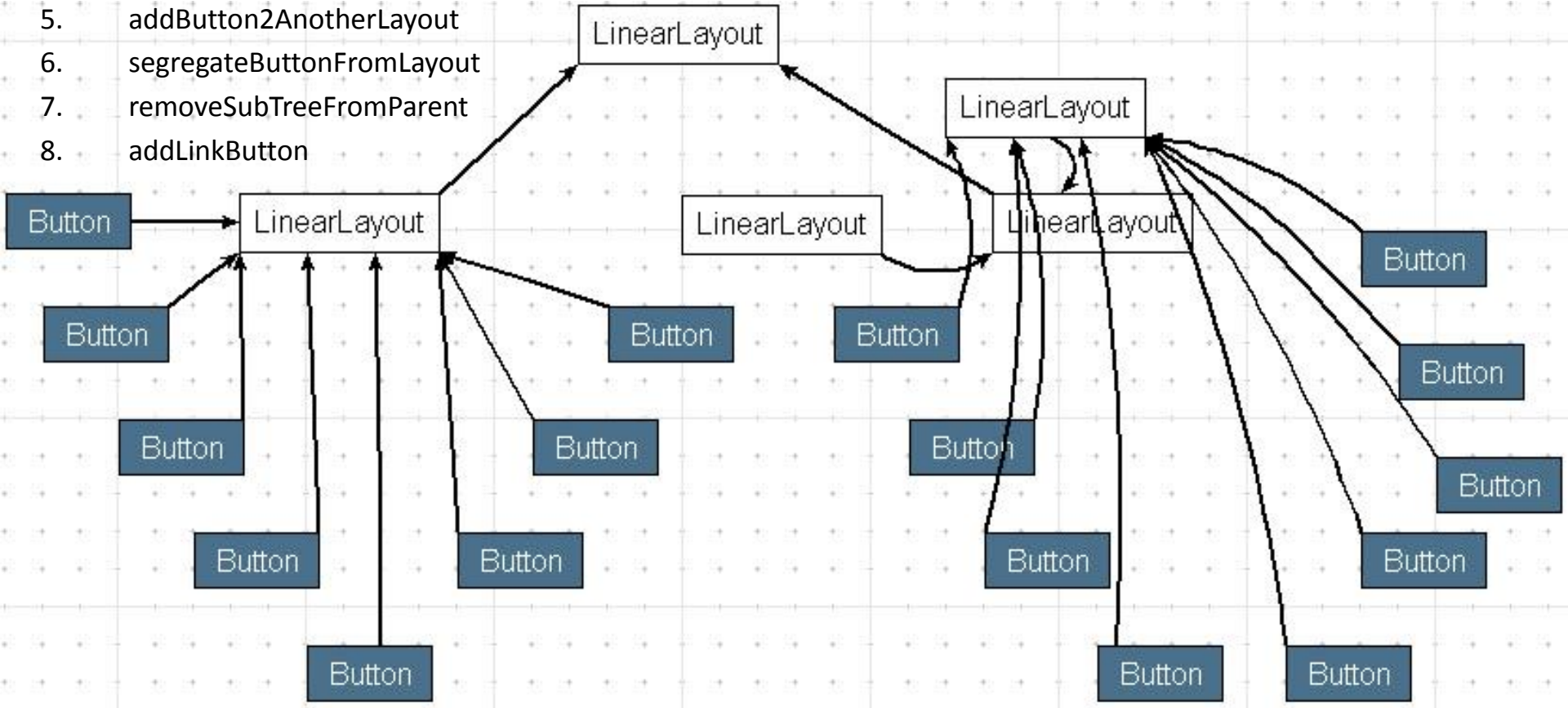
# Horizontal Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



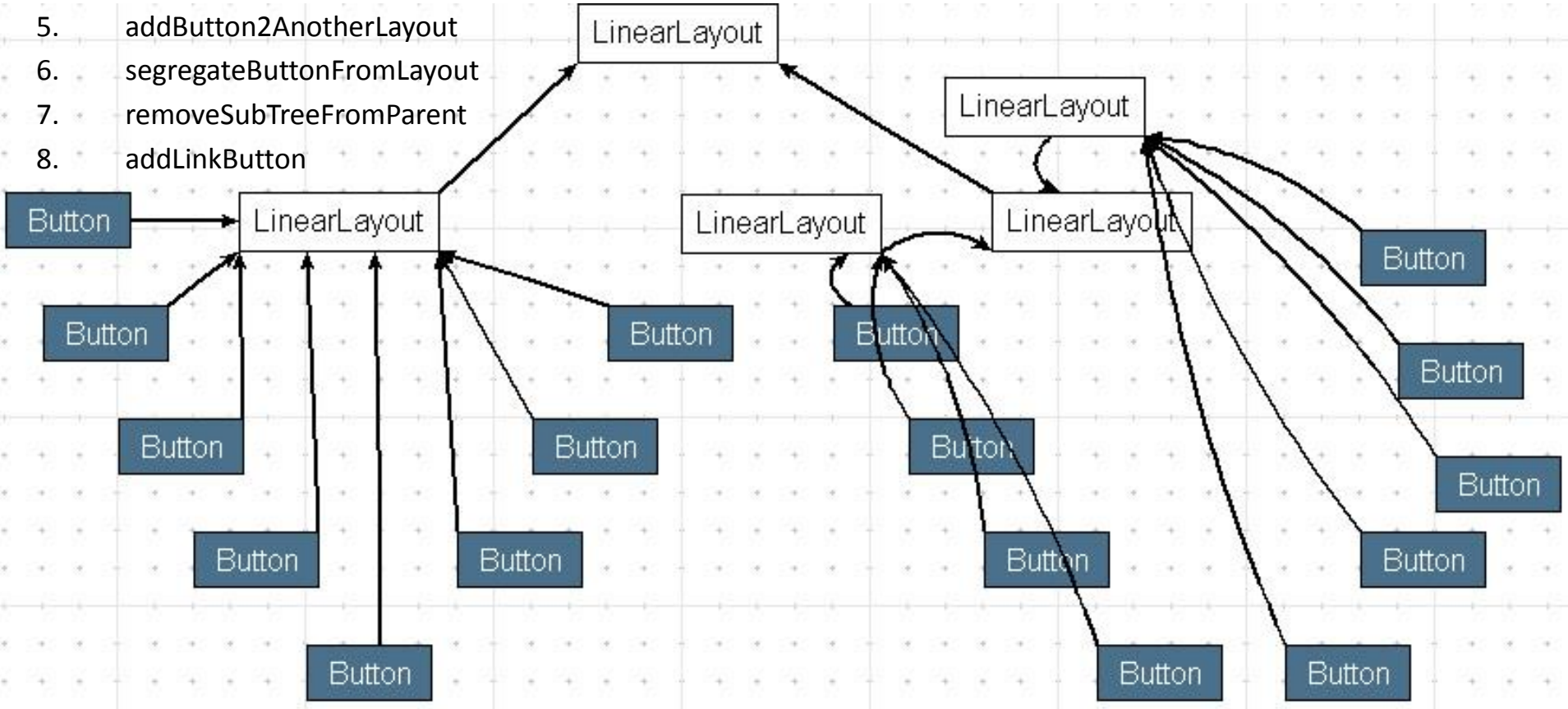
# Horizontal Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



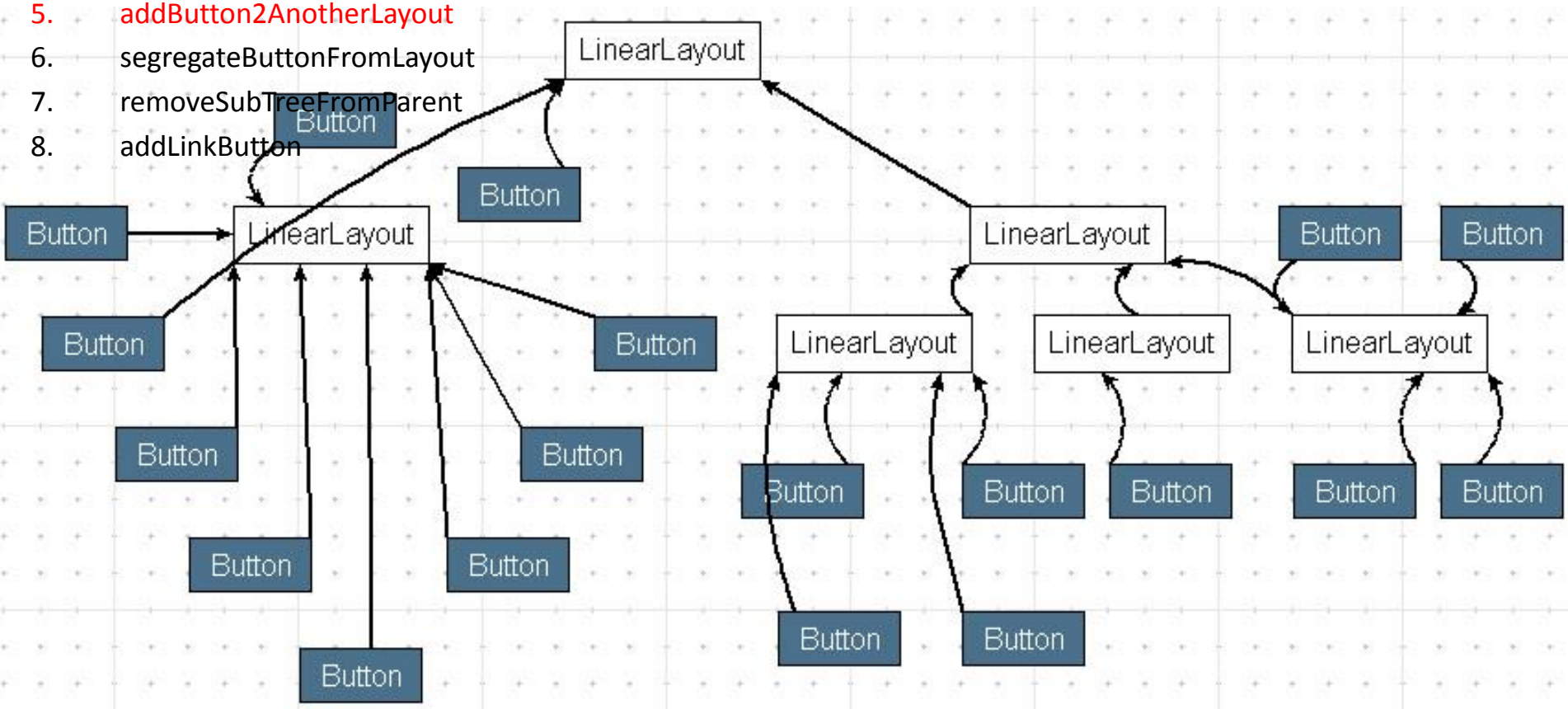
# Horizontal Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



# Vertical Transformation

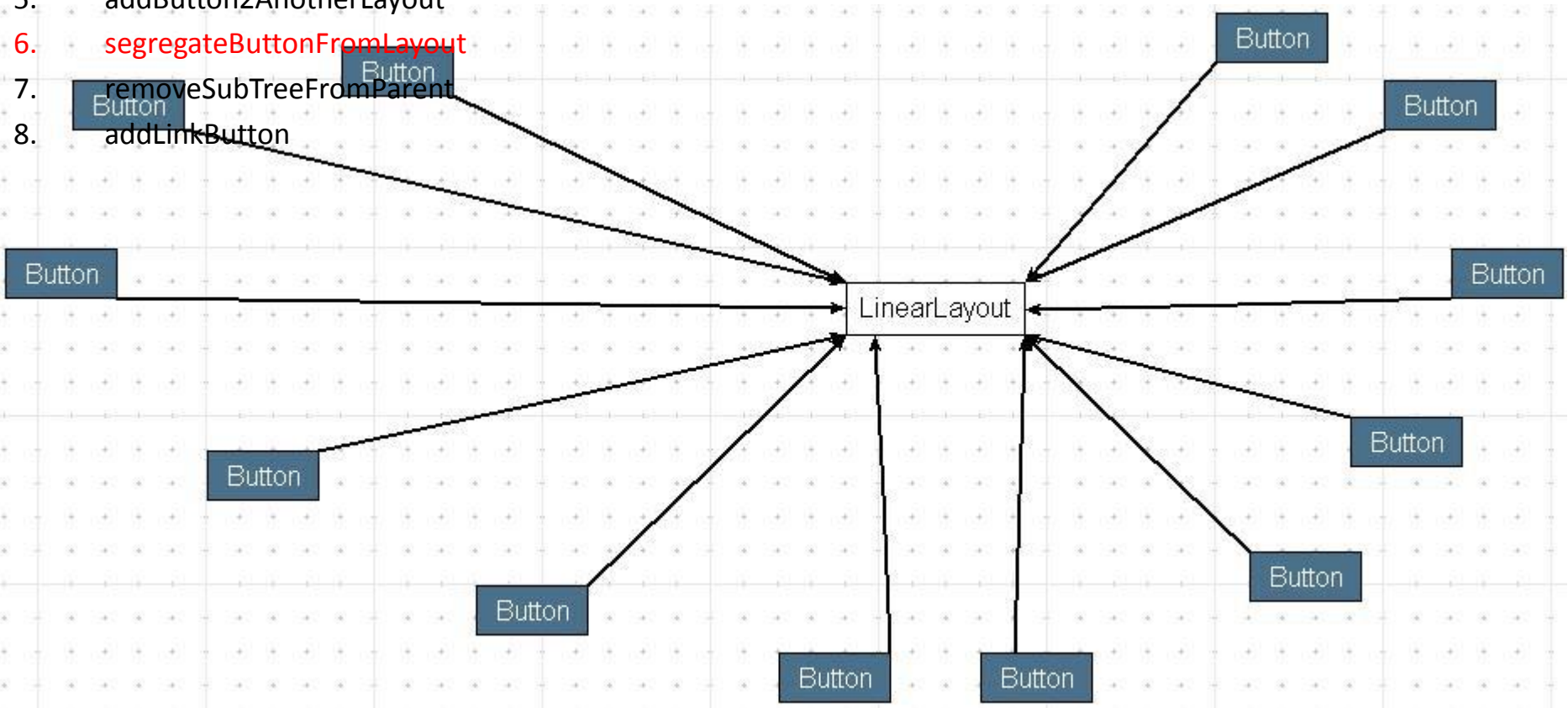
1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



# Vertical Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton

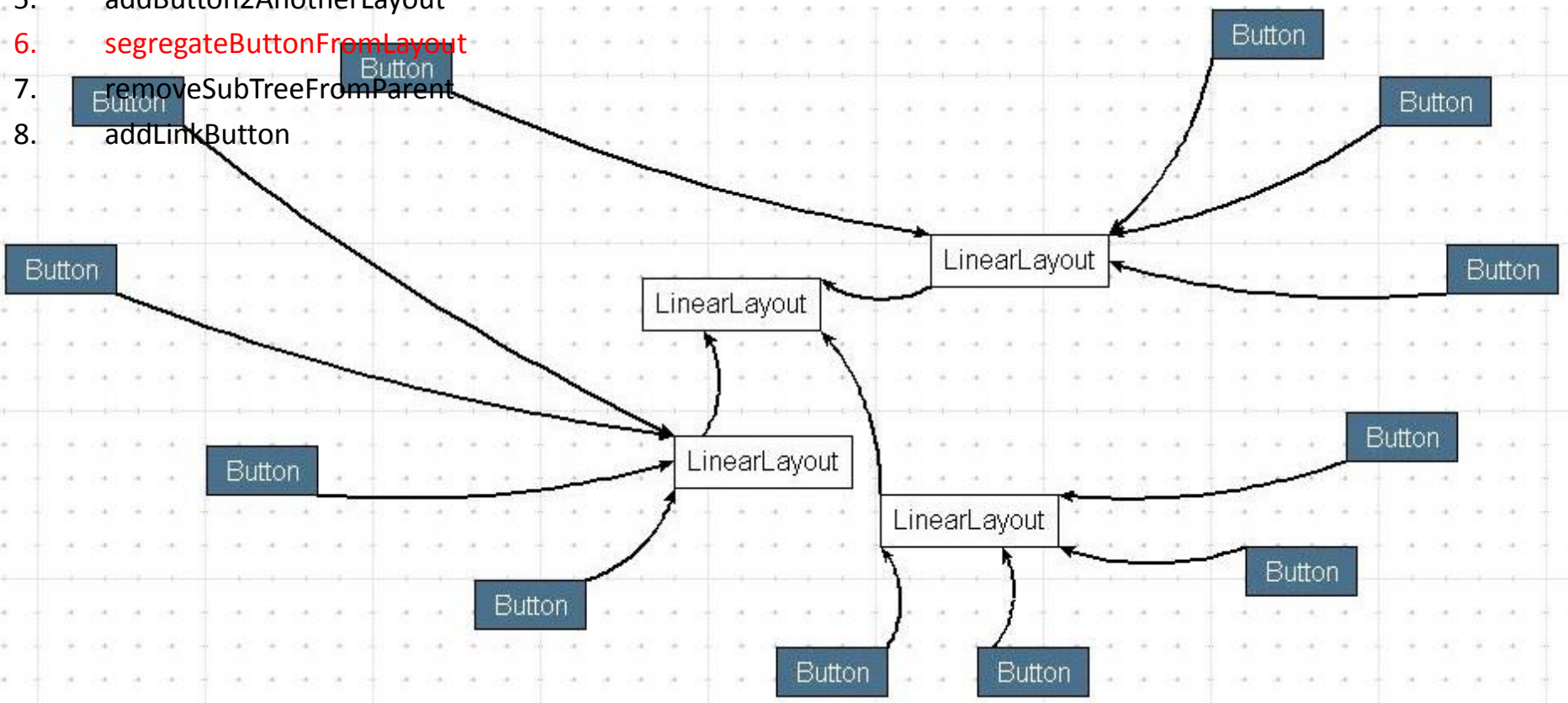
New example



# Vertical Transformation

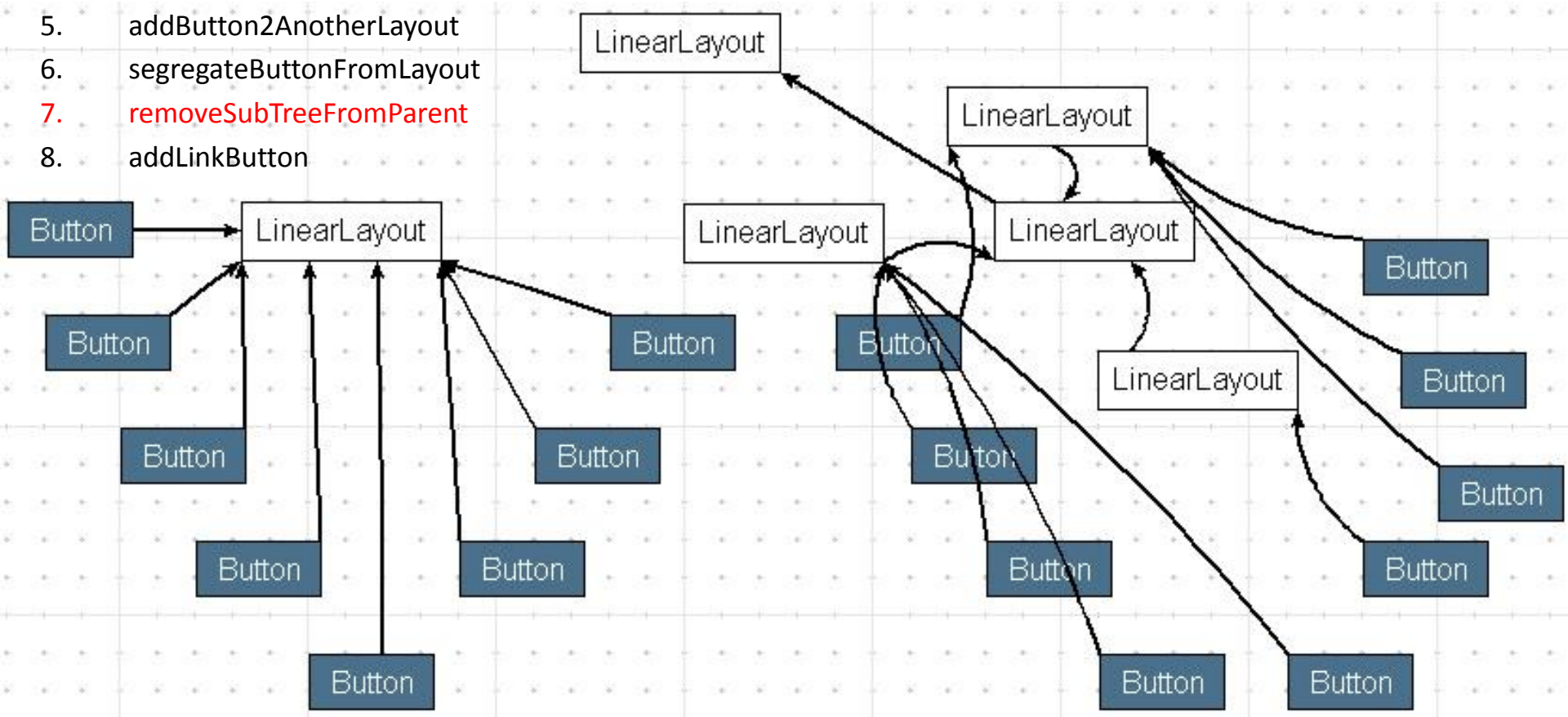
1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton

New example



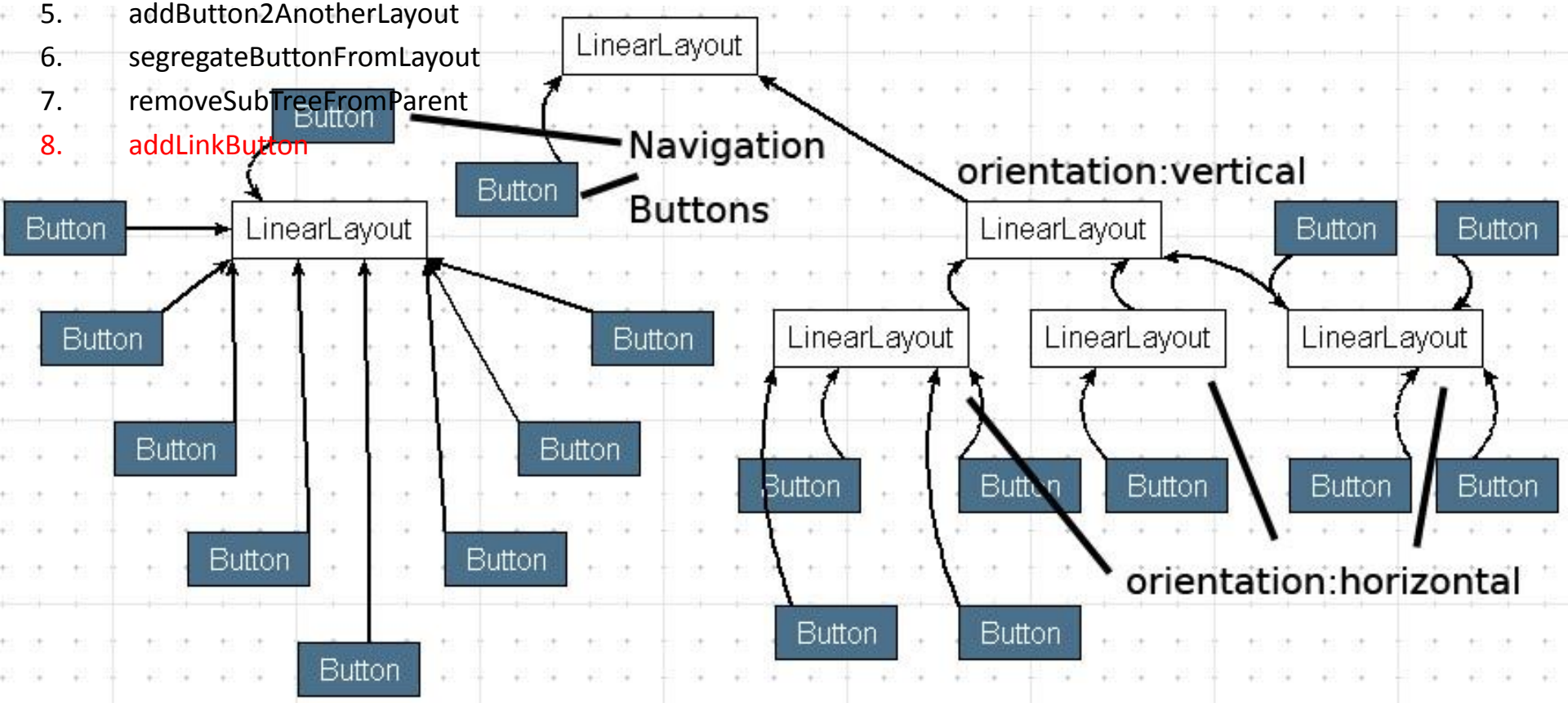
# Vertical Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



# Vertical Transformation

1. setTopNodeAndNewBranch
2. moveButton2NewBranch
3. setNewBranchWhenFull
4. moveExcessButton2NewBranch
5. addButton2AnotherLayout
6. segregateButtonFromLayout
7. removeSubTreeFromParent
8. addLinkButton



# Outline

- Introduction
- Problems analysis
- From AndroidDevice to AndroidGUI
- From unconstrained to constrained
- From model to files
- Future work
- References

# AndroidManifest.xml

- To register all the activities/layouts

```
#generate the androidmanifest file
i = 0
firstPage = True
for key in self.ASGroot.listNodes.keys():
  if self.ASGroot.listNodes[key] != []:
    output = open('AndroidManifest.xml', "w")
    output.write("<?xml version="+'"1.0"'+" encoding="+'"utf-8"'+'?>'+'\n')
    output.write("<manifest xmlns:android=")
    output.write('"http://schemas.android.com/apk/res/android"'+'\n')
    output.write('package='+'"yuan.android.translucent"'+'>\n')
    output.write("<application android:icon="+'"@drawable/icon"'+'>\n')
    for node in self.ASGroot.listNodes[key]:
      if node.out_connections_ == []:
        if firstPage:
          generateAndroidManifest(output, firstPage, i)
          firstPage = False
        else:
          print 'print branch layouts now'
          generateAndroidManifest(output, firstPage, i)
        i += 1
    output.write('</application>'+'\n'+'</manifest>')
    output.close()
    print "successfully write to AndroidManifest.xml"

def generateAndroidManifest(outfile, isFirstPage, number):
  if isFirstPage:
    print str(isFirstPage)
    outfile.write('<activity'+" android:name="+'".AndroidPage'+str(number)+'"'+" android:label="+'"@string/app_name"'+'>\n')
    outfile.write('<intent-filter>'+'\n')
    outfile.write("<action android:name="+'"android.intent.action.MAIN"/>'+'\n')
    outfile.write("<category android:name="+'"android.intent.category.LAUNCHER"/>'+'\n')
    outfile.write('</intent-filter>'+'\n')
    outfile.write('</activity>'+'\n')
  else:
    print str(isFirstPage)
    outfile.write('<activity'+" android:name="+'".AndroidPage'+str(number)+'"'+'>\n')
    outfile.write('</activity>'+'\n')
```

Key point:

1. Find every root LinearLayout node for the layout (Exp: AToM3 terms)
2. Traverse nodes on the graph
3. Determine root node by its out_connections_

# User Interface XML files

- A UI XML file for a layout

```
#generate ui xml files
for key in self.ASGroot.listNodes.keys():
  if self.ASGroot.listNodes[key] != []:
    for node in self.ASGroot.listNodes[key]:
      if node.out_connections_ == []:
        output = open('page'+str(i)+'.xml', "w")
        output.write("<?xml version="+'"1.0"'+" encoding="+'"utf-8"'+'?>'+'\n')
        generate(node, output, True)
        output.close()
        print "save to file "+'page'+str(i)+'.xml'+' successfully!'
        i += 1
```

```
import globalVariables

def generate(node, outfile, isFirstNode):
 if (node.getClass() == 'AButton'):
  outfile.write('<'+'Button'+'\n')
 else:
  outfile.write('<'+node.getClass()+'\n')
 if (isFirstNode):
  outfile.write('xmlns:android='+'"http://schemas.android.com/apk/res/android"'+'\n')
 for attr in node.realOrder:
  outfile.write('android:'+attr+'="'+str(node.getAttrValue(attr).getValue())+'"\n')
 outfile.write('>'+'\n')
 if (node.in_connections_):
  for association in node.in_connections_:
   for childnode in association.in_connections_:
    generate(childnode, outfile, False)
 if (node.getClass() == 'AButton'):
  outfile.write('</'+'Button'+'>\n')
 else:
  outfile.write('</'+node.getClass()+'>\n')
```

Key point:

1. Find every instantiated node on the graph
2. Insert them in a tree-like structure
3. Iteration from a high level node to low level nodes

# Java/Android source files

- A source file for every layout
- UI XML files for UI design
- Source files for the behaviour
- Navigation button

```
#generate android/java code files
i = 0
j = 0
for key in self.ASGroot.listNodes.keys():
  if self.ASGroot.listNodes[key] != []:
    for node in self.ASGroot.listNodes[key]:
      if node.out_connections_ == []:
        j += 1

for key in self.ASGroot.listNodes.keys():
  if self.ASGroot.listNodes[key] != []:
    for node in self.ASGroot.listNodes[key]:
      if node.out_connections_ == []:
        globalVariables.firstButton = True
        globalVariables.firstTextView = True
        globalVariables.firstEditText = True
        globalVariables.firstSpinner = True
        globalVariables.firstProgressBar = True
        globalVariables.firstCheckBox = True
        globalVariables.firstLinearLayout = True
        output = open('AndroidPage'+str(i)+'.java', "w")
        output.write("package yuan.android.translucent;"+'\n')
        output.write(''+'\n')
        output.write("import android.app.Activity;"+'\n')
        output.write("import android.os.Bundle;"+'\n')
        generateAndroidCode(node, output)
        output.write(''+'\n')
        if node.in_connections_ != []:
          for assoc in node.in_connections_:
            for subNode in assoc.in_connections_:
              if hasattr(subNode, "text"):
                if subNode.text.getValue() == 'linkButton':
                  output.write("import android.content.Intent;\n")
        output.write(''+'\n')
        output.write("public class "+'AndroidPage'+str(i)+" extends Activity {"+'\n')
        if node.in_connections_ != []:
          for assoc in node.in_connections_:
            for subNode in assoc.in_connections_:
              if hasattr(subNode, "text"):
                output.write("    private Button linkButton;"+'\n')
        output.write("    @override"+'\n')
        output.write("    public void onCreate(Bundle icicle) {"+'\n')
        output.write("        super.onCreate(icicle);"+'\n')
        output.write("        this.setTheme(R.style.Theme_Translucent);"+'\n')
        output.write("        setContentView(R.layout."+'page'+str(i)+');'+'\n')
        if node.in_connections_ != []:
          for assoc in node.in_connections_:
            for subNode in assoc.in_connections_:
              if hasattr(subNode, "text"):
                if subNode.text.getValue() == 'linkButton':
                  output.write("        linkButton = (Button) findViewById(R.id.linkButton);"+'\n')
                  output.write("        linkButton.setOnClickListener(listener);"+'\n')
        output.write("    }"+'\n')
        output.write(''+'\n')
        if node.in_connections_ != []:
          for assoc in node.in_connections_:
            for subNode in assoc.in_connections_:
              if hasattr(subNode, "text"):
                if subNode.text.getValue() == 'linkButton':
                  output.write("    private OnClickListener listener = new OnClickListener() {\n")
                  output.write("        @Override\n")
                  output.write("        public void onClick(View view) {"+'\n')
                  if (i+2) < j or (i+2) == j:
                    print "inside first selection"
                    print "i: "+str(i)
                    print "j: "+str(j)
                    output.write("            Intent intent = new intent(this, "+'AndroidPage'+str(i+1)+'.class);\n')
                  else:
                    print "inside second selection"
                    print "i: "+str(i)
                    print "j: "+str(j)
                    output.write("            Intent intent = new intent(this, "+'AndroidPage'+str(i-1)+'.class);\n')
                  output.write("            intent.setLaunchFlags(Intent.NEW_TASK_LAUNCH);\n")
                  output.write("            startActivity(intent);\n")
                  output.write("        }\n")
                  output.write("    };\n")
        output.write('}\n')
        output.close()
        print "save to file successfully!"
        i += 1
```

```
def generateAndroidCode(node, outfile):
  if globalVariables.firstLinearLayout:
    outfile.write("import android.widget.LinearLayout"+'\n')
    globalVariables.firstLinearLayout = False
  if node.in_connections_ != []:
    for assoc in node.in_connections_:
      for subNode in assoc.in_connections_:
        if (hasattr(subNode, "orientation")):
          generateAndroidCode(subNode, outfile)
        else:
          if subNode.getClass() == 'AButton' and globalVariables.firstButton:
              outfile.write("import android.widget.Button;"+'\n')
              globalVariables.firstButton = False;
          elif subNode.getClass() == 'TextView' and globalVariables.firstTextView:
              outfile.write("import android.widget.TextView;"+'\n')
              globalVariables.firstTextView = False;
          elif subNode.getClass() == 'EditText' and globalVariables.firstEditText:
              outfile.write("import android.widget.EditText;"+'\n')
              globalVariables.firstEditText = False;
          elif subNode.getClass() == 'Spinner' and globalVariables.firstSpinner:
              outfile.write("import android.widget.Spinner;"+'\n')
              globalVariables.firstSpinner = False;
          elif subNode.getClass() == 'ProgressBar' and globalVariables.firstProgressBar:
              outfile.write("import android.widget.ProgressBar;"+'\n')
              globalVariables.firstProgressBar = False;
          elif subNode.getClass() == 'CheckBox' and globalVariables.firstCheckBox:
              outfile.write("import android.widget.CheckBox;"+'\n')
              globalVariables.firstCheckBox = False;
```

# Future work

- Better modeling environment for Android UI design
- Intelligent distribution of components
- Consistent User Interface

# References


- [1] Jeffrey Nichols et al. “Generating Remote Control Interfaces for Complex Appliances”, UIST 2002
- [2] Brouwe-Janese et al. “Interfaces for consumer products: “how to camouflage the computer?”” CHI 1992
- [3] Jeffrey Nichols et al. “UNIFORM: Automatically Generating Consistent Remote Control User Interfaces” CHI 2006

# Thank you!