Model-driven Reengineering for a Blue Planet
- Refactoring for Energy Efficiency -

Andreas Winter

joint work with
Marion Gottschalk
Jan Jelschen
Mirco Josefiok
"Energy Turnaround"

not far away from here

online: March 1, 1986 – Sept. 9, 1988

April 26, 1986

March 11, 2011

© Andreas Winter 11.10.2012 SoTeSoLa 2012
"Energy Turnaround"

2020: have 35% green power
2022: switch off all nuclear power plants

online: March 1, 1986 – Sept. 9, 1988

March 11, 2011
How will

Software Engineering
Software Language Engineering
Model-driven Software Reengineering

help to manage the

energy turnover
Green Software Engineering

Current Situation
- ICT consumes more than 10% of Germany's energy budget (2007)
- ICT produces more CO$_2$ emissions than the entire German aviation sector

Research in Computer Science
- focusses on saving energy
  - in computing centers
  - on operating system level
  - on hardware level
- does not consider software applications

Applying Software-Reengineering
- analyzing code for wasting energy
- improving energy inefficient code
- providing information on user behavior to enable more energy efficient task scheduling by OS

discovering waste of energy early
avoiding waste of energy by energy aware computing
controlling energy consumption of hardware components by software
Reengineering = Quality Improvement

Static Analysis
- define and detect “energy code smells”
- define metrics for energy-efficiency classification

Dynamic Analysis
- log application behavior and correlate with battery drain
- monitor user behavior and deduct usage patterns
- detect periodic wake-ups and synchronize

Refactoring
- replace energy-inefficient code
- introduce strategy patterns to enable apps to make energy-aware choices
Refactoring = Identification + Restructuring

Energy Code Smells
- depict energy-inefficient patterns in code

Process
1. Preparation:
   - parse source to TGraphs
2. Identification:
   - identify code patterns by graph queries (GReQL)
3. Restructuring:
   - improve code through graph transformations
4. Post processing:
   - unpars TGraphs to source

Reengineering Architecture
Example: GPSPrint

Potential waste of energy
- energy inefficient use of GPS sensor

[http://play.google.com/store/apps]
Resource Utilization in Android

Example: Binding resources too early

Identification

- Where is the GPS sensor turned on?
- Where does the `locationManager` call `requestLocationUpdates()` for the GPS?

Bad energy smell

- Call of `requestLocationUpdates()` in `onCreate()`
Metamodel

Mission
- representing Java code for efficient querying
- fine grained Java representation
  - contains
    - 86 node types
    - 67 edge types

GPSPrint Graph
- contains
  - 9034 nodes
  - 14880 edges
Identification

finding the smelling call
of `requestLocationUpdates()`
Restructuring

- identify energy code smell
- identify code to be refactored
- do the refactoring

```java
public class GpsPrint extends Activity
  implements OnClickLister, Listener, LocationListener {

  [...] public void onCreate(Bundle
    savedInstanceState) {

  [...] LocationManager lm=(LocationManager)
    getSystemService(Context.
    LOCATION_SERVICE);
  if(lm.getAllProviders().contains(
    LocationManager.GPS_PROVIDER)) {
    if(lm.isProviderEnabled(
      LocationManager.GPS_PROVIDER)) {
      lm.addGpsStatusListener(this);
      lm.requestLocationUpdates(LocationManager.
        GPS_PROVIDER, 1000, 0, this);
      status_view.setText(
        "GPS service started");
    } else {
      status_view.setText(
        "Please enable GPS");
      save_location_button.setEnabled(
        false);
    }

  [...] }

  [...] public void on_pause() {

  [...] lm.removeUpdates(this);

  [...] public void onResume() {

  [...] lm.requestLocationUpdates(
    LocationManager.GPS_PROVIDER,
    1000, 0, this);

  [...] }

  Before Refactoring
```
Restructuring – Graph Transformation

```
<table>
<thead>
<tr>
<th>v8676</th>
<th>frontend.java.DataObject</th>
</tr>
</thead>
<tbody>
<tr>
<td>fullyQualifiedName = &quot;android.app.Activity&quot;</td>
<td></td>
</tr>
<tr>
<td>name = &quot;Activity&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e5563</th>
<th>frontend.java.HasSuperClass</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3488</td>
<td>frontend.java.Class</td>
</tr>
<tr>
<td>name = &quot;GpsPrint&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e6655</th>
<th>frontend.java.HasMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3643</td>
<td>frontend.java.DataObject</td>
</tr>
<tr>
<td>name = &quot;onCreate&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e6654</th>
<th>frontend.java.DataObjectHasType</th>
</tr>
</thead>
<tbody>
<tr>
<td>v3644</td>
<td>frontend.java.MethodType</td>
</tr>
<tr>
<td>name = &quot;onCreate&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e10572</th>
<th>frontend.java.DataObjectHasType</th>
</tr>
</thead>
<tbody>
<tr>
<td>v6212</td>
<td>frontend.java.MethodType</td>
</tr>
<tr>
<td>name = &quot;onResume&quot;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>e10467</th>
<th>frontend.java.ext.CallsMethod</th>
</tr>
</thead>
<tbody>
<tr>
<td>v8823</td>
<td>frontend.java.DataObject</td>
</tr>
<tr>
<td>name = &quot;requestLocationUpdates&quot;</td>
<td></td>
</tr>
</tbody>
</table>
```
Restructuring

Graph Transformation
  - using JGraLab

```java
public class DeleteEdgeGPSPrint extends Transformation<Object> {
    [...] 
    @Override
    protected Object transform() {
        OpenSaveTG tg = new OpenSaveTG();
        // loads TGraph (see figure 3)
        TGraph = tg.openTG("GPSPrint-pruned07.tg");
        try {
            // searches for super class of 'GpsPrint'-class
            [...] 
            // searches for MethodType 'onCreate' which calls
            the dataObject 'requestLocationUpdates'
            for (Vertex v : TGraph.vertices()) {
                if (v.getAttribute("name").equals("onCreate") &&
                    (v.isBefore(superClass) || v.isAfter(
                        superClass))) {
                    for (Edge e : v.incidences()) {
                        // considers only out-going edges
                        if (e.getId() > 0) {
                            if (e.getThat().getAttribute("name").equals("requestLocationUpdates")) {
                                // deletes unnecessary edge
                                e.delete();
                                tg.saveTG("GPSPrintModified");
                                return "";
                            }
                        }
                    }
                }
            }
        }
    }
}
```
Further Classes of Energy Code Smells

Loop Bug
- application is repeating the same activity

Dead Code
- source code which is never used, but loaded to memory

In-line method
- replacing method calls by the body of the called method

Moving too much data
- unnecessary communication between processor and memory

Immortality Bug
- applications respawning after explicitly being killed by the user

Types of Energy Refactorings
- classical refactoring (Fowler)
- general energy refactorings
- platform specific

Types of Energy Analysis
- static analysis
- dynamic analysis

Remark
- energy refactorings change behavior of apps

Redundant storage of data
- different methods of store the same data in memory

Using expensive resources
- energy-expensive resources are exchange to “cheaper” alternatives (algorithms and hardware components)
Model driven Reengineering is applicable to improve energy-behavior of software

- using TGraphs for representation
- applying GReQL for querying
- utilizing JGraLab for code restructurings

Further investigation on energy code smells

- finding and defining more energy code smell classes and their associated refactorings
- providing an platform independent means for energy quantification
  - to prove benefits of energy code smell refactorings