Correct deployment and adaptation of software applications on heterogeneous (mobile) devices

F. Mancinelli, P. Inverardi, G. Marinelli
{mancinell, inverard, gmarinel}@di.univaq.it

SEA Group
Dipartimento di Informatica
Università dell’Aquila
Setting the context

- Heterogeneous (mobile) devices
  - Same basic functionalities
  - Different quantitative and qualitative characteristics

- Possibly infinite device characteristics (screen size, memory size, power, communication protocols, etc.)

- Check application compatibility with respect to a given set of characteristics and perform adaptation in order to prevent runtime execution failures
Setting the context

- Formal framework based on an approach to develop and distribute adaptable applications

- Ideas borrowed from Proof Carrying Code (PCC) [Necula, 97]

- Chosen reference platform is Java 2 MicroEdition with the MIDP Profile

- Assumptions:
  - Target devices are limited
  - Tailored adaptable applications (instead of self contained adaptable applications)
  - Device Functionalities are characterizable in a discrete way
  - Applications are relatively small and not so much complex
Framework architecture

Client

Safety Policy

Tailored application

Server

Development tools

Annotated Application code

VCGen

Safety Predicate

Theorem Prover

Customizer
Framework approach characteristics

- Static approach which captures some dynamic properties
- Best fit approach
- Lightweight with respect to the client
- Formal
- Declarative approach to manage qualitative properties
Framework approach

n **Step1:** Annotated source code development, definition of an adaptation policy and source code compilation

n **Step2:** Safety predicate generation

n **Step3:** Proof generation

n **Step4:** Construction of the final adapted code
A case study: the screen

- Different devices
- Different screen capabilities
- Same application with different (possibly incorrect or undesired) behaviours
Step 1
Annotated source code development

n Standard annotations
  ▷ Loop/branches invariants

n Adaptation policy:
  ▷ Adaptation points
  ▷ Adaptation alternatives (for each adaptation point)

n Syntactical construct:
  \[ \text{ADAPT} \{ c_1 \} \]
  \[ \text{USE} \{ c_2 \} \]
  ...
  \[ \text{USE} \{ c_n \} \]
Step 1
Annotated Java source code

```java
public void paint(Graphics g) {
    int x; int y;
    x = 10; y = 50;
    g.drawRect(0, 0, subtract(x, y), 50);
    ADAPT { g.drawRect(0, 0, 120, 10); }
    USE { g.drawRect(0, 0, 50, 10); }
    USE { g.drawRect(0, 0, 10, 10); }
}

public int subtract(int x, int y) {
    if(x < y) return 0;
    return (x - y);
}
```

Code compilation produces an annotated relocatable byte code

Code compilation should ensure the type correctness of each program version derived using the adaptation policy
Step 1
Relocatable annotated bytecode

Method void paint(Graphics g)
  0 bipush 10
  2 istore_1
  3 bipush 50
  5 istore_2
  6 aload_0
  7 iconst_0
  8 iconst_0
  9 aload_0
 10 iload_1
11 iload_2
12 invokevirtual #2 <Method int
    subtract(int, int)> |
  15 bipush 50
  17 invokevirtual #3 <Method void
    rect(int, int, int, int)> |
  20 ADAPT1(b1, b2, b3)
 20+11 return

Method int subtract(int, int)
  0 iload_1
  1 iload_2
  2 if_cmpge 7
  5 iconst_0
  6 ireturn
  7 iload_1
  8 iload_2
  9 isub
10 ireturn
Step 2

- Given the annotated relocatable byte code and a safety policy, the safety predicate is built by the VCGen.

- Adaptation policy alternatives are transparently embedded in the safety predicate.
Step 2
The safety predicate

\[ \text{RECT}(x, y, z, w) \]

\[ \Rightarrow \]

Safety policy (provided by the client)

Predicate obtained from the annotated bytecode

\[ (10 \geq 50 \Rightarrow \text{RECT}(0, 0, (10-50), 50) : \text{Visible} \land \\
10 < 50 \Rightarrow \text{RECT}(0, 0, 0, 50) : \text{Visible}) \]

\[ \land \]

\[ \text{OR(RECT}(0, 0, 120, 10) : \text{Visible}, \text{RECT}(0, 0, 50, 10) : \text{Visible}, \text{RECT}(0, 0, 10, 10) : \text{Visible}) \]
Step 3
The proof system

Proof system:
- Proof rules (FOL, Properties specific)
- Proof Algorithm

It must be decidable and modular

Proof $\Rightarrow$ Configuration
Step 3
The proof

P_1 = P_2 \land P_7
P_2 = 10 \leq 50 \Rightarrow \text{RECT}(0, 0, (10-50), 50):\text{Visible} \land 10 < 50 \Rightarrow \text{RECT}(0, 0, 0, 50):\text{Visible}
P_3 = 10 < 50 \Rightarrow \text{RECT}(0, 0, 0, 50):\text{Visible}
P_4 = \text{RECT}(0, 0, 0, 50):\text{Visible}
P_5 = 10 \geq 50 \Rightarrow \text{RECT}(0, 0, (10-50), 50):\text{Visible}
P_6 = 10 \geq 50
P_7 = \text{OR}(\text{RECT}(0, 0, 120, 10):\text{Visible}, \text{RECT}(0, 0, 50, 10):\text{Visible}, \text{RECT}(0, 0, 10, 10):\text{Visible})
Step4
Tailored application

Method void paint(Graphics g)
0 bipush 10
2 istore_1
3 bipush 50
5 istore_2
6 aload_0
7 iconst_0
8 iconst_0
9 aload_0
10 iload_1
11 iload_2
12 invokevirtual #2 <Method int subtract(int, int)>
15 bipush 50
17 invokevirtual #3 <Method void rect(int, int, int, int)>
20 aload_0
21 iconst_0
22 iconst_0
23 bipush 50
25 bipush 10
27 invokevirtual #3 <Method void rect(int, int, int, int)>
30 return
Conclusions and future works

- Effectiveness of a declarative approach
- The approach is thought to have little impact on the devices
- We are extending the adaptation with respect to other characteristics
- Implement all the tools needed by the framework (compilers, ad-hoc theorem prover...)