Refactoring the Java Card Framework

Geraud Campion, Michael O'Connor
Faculty Advisor: Steve Zdancewic, Alwyn Goodloe

Abstract

• Our project involves Smart Cards. Smart Cards (SC) have a small CPU and a limited amount of memory; they communicate with a terminal specifically programmed to interact with a SC. SC and Terminal programs are developed in Java; however, SCs use a language that is a subset of Java.

• Our project will take in a single annotated Java program with both Terminal code and SC code written in the same file. It will then create two separate files: one file holding card code with methods translated so that it may call the terminal, the other file being terminal code.

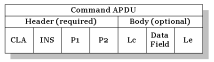
Before:

```
public class Card{
  public void
Process(){}
public short getX(){}
public boolean getY()
{}
}
```

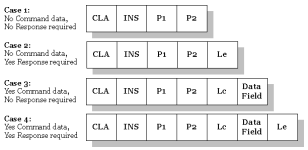
```
public class Term{
public void foo(){}
public int getX(){}
public boolean getY()
{}
}
```

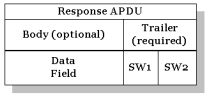
After:

```
public class Combined{
  @TERM private int x;
  @CARD private short x;
  @BOTH private boolean y;
  @CARD public void Process(){}

  @TERM public void foo() {}

  @CARD public short getX();
  @TERM public int getX();
  @BOTH public boolean getY();
}
```

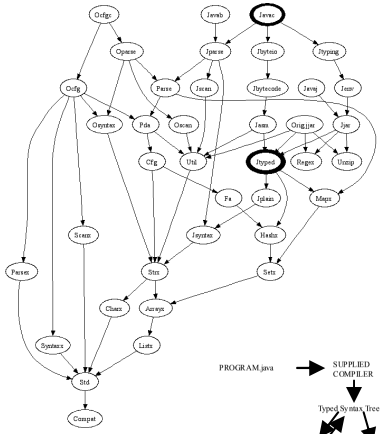


Possible Formats of the Command APDU.

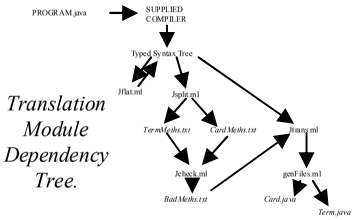




Format of the Response APDU.



Java Compiler Module Dependency Tree



Translation Module Dependency Tree.

```
class Scheduler {

  Object obj;
  short  s;
  ... ...
  public static void goto(Object o, short s ) {
     obj = o; state = s;
  }
  ... ...
  public static void mainloop() {
    ... ...
    while (true) {
      switch (state) {
      STATE_sendAndRecv:        return;
      ........
      STATE_authenticate_entry: (MyApp)obj.authenticate_entry();
      STATE_5:                  (MyApp)obj.line_5();
      STATE_6A:                 (MyApp)obj.line_6A()
      STATE_6B:                 (MyApp)obj.line_6B();
      STATE_7:                  (MyApp)obj.line_7();
      STATE_9:                  (MyApp)obj.line_9();
      STATE_10:                 (MyApp)obj.line_10();
      STATE_11:                 (MyApp)obj.line_11();
      STATE_authenticate_exp:   (MyApp)obj.authenticate_exp();
      ........
```

The Scheduler's generated switch states

$$
\begin{aligned}
&\ldots\ldots && \text{where} && \ldots \\
&\mathbb{S}_{\mathsf{stmt}}[\![s_m]\!]_{\eta_m} && && \eta_m = (t_{m-1}, n, x)
\end{aligned}
$$

$$
\begin{aligned}
&\mathbb{S}_{\mathsf{stmt}}[\![\mathsf{if}(e)\ s_1\ \mathsf{else}\ s_2]\!]_\eta \\
&= \mathbb{B}_{\mathsf{branch}}(\mathbb{S}_{\mathsf{exp}}[\![e]\!], t_1, t_2)_\eta && && \eta = (b, n, x) \\
&\quad \mathbb{S}_{\mathsf{stmt}}[\![s_1]\!]_{\eta_1} && \text{where} && \eta_1 = (t_1, n, x) \\
&\quad \mathbb{S}_{\mathsf{stmt}}[\![s_2]\!]_{\eta_2} && && \eta_2 = (t_2, n, x)
\end{aligned}
$$

$$
\begin{aligned}
&\mathbb{S}_{\mathsf{stmt}}[\![\mathsf{while}(e)\ s]\!]_\eta \\
&= \mathbb{B}_{\mathsf{branch}}(\mathbb{S}_{\mathsf{exp}}[\![e]\!], t_1, n)_\eta && \text{where} && \eta = (b, n, x) \\
&\quad \mathbb{S}_{\mathsf{stmt}}[\![s]\!]_{\eta_1} && && \eta_1 = (t_1, n, x)
\end{aligned}
$$

A few of the translation algors.

The resulting translated code

```
public class Card{
  private int x, y;
  public void Process()
{... }
  public short getX()
{... }
public boolean getY()
{... }
}
```

```
public class Term{
private int x, y;
  public void foo()
{... }
  public int getX()
{... }
public boolean getY()
{... }
}
```

Conclusion

We initially created a Smart Card and Terminal project, and learned the problems with the master-slave model that we tried to alleviate with the second phase of our project. In the second half of our project, we actually created this translation of a SC program. Here we learned Ocaml, and functional programming, which was a great challenge for us. After we cleared this hurdle, we had to dissect and learn the compiler and environment that we were going to build off. Then, we were finally able to work on our translator. First, we created a Jflat program to flatten a given statement or expression. Then we implemented the paper's translation of SC code; however, we did not translate the terminal code. In the end, we created a program that takes in an annotated Java program, creates two separate programs from this one, which is fully translated to break the master-slave model that it was previously tied to.