

Variability Models

Dave Clarke

June 13, 2014

Abstract

This document includes a partial specification of a drink machine product line, a description of the tasks to be performed during the course, and cheat sheets for various formalisms used.

1 Drink Machine Product Line Specification

The Drink Machine Product Line consists of the following features.

Features: Tea, Coffee

Different drink options are available: tea, coffee. Tea is essentially just hot water. No milk or sugar is provided. Click button and drink will be served, assuming that payment has been made (if required).

Feature: Sugar and Milk

Tea and coffee now have different options for milk and sugar. Additional buttons will be provided to add different amounts of sugar and milk.

Feature: Fancy Coffee

A wider selection of coffees — cappuchino, latte, espresso, machiato, hot chocolate — are available. A new panel will offer a choice between these drinks.

Feature: Cup Dispenser

The machine can provide cups, require/allow user to bring own cup (whose presence the machine detects), or both. When cups run out, cups are no longer available (obviously). In this case, a signal can be sent to display.

Feature: Primitive Display

Display consists of a board of 3 LEDs (red, green, yellow) which can be turned on or off programmatically, for instance, to indicate that a drink is being served or that ingredients have run out.

Feature: Advanced Display

A small LCD display is available. Any text can be programmatically displayed on this. Display can also be cleared programmatically.

Features: Cash payment, Card payment, No payment

These features would allow the machine to accept cash, cards, or give free drinks. For simplicity, assume all drinks cost 1 coin. Payment is made and change given before drink is delivered.

Feature: Refill

Drinks/ingredients can run out. The refill features monitors these and can use display to show status messages. Refill functionality is available—refilling is possible before ingredients run out.

Feature: Cancellation

Cancel purchase. Can take place before drink is delivered. Resets system to initial state (except for ingredients used) and refunds coins. If card payment was being used, cannot cancel after payment transaction has begun.

Additional Constraints

Assume that all drinks use 1 unit of whatever ingredient it needs, or multiples in the case of sugar and milk.

2 Tasks

1. Develop Feature Model

- Identify all features
- Identify interactions, conflicts and dependencies between features
- Design a feature model to capture the variability and rule out conflicts — if required, use abstract features to structure model better, and try to minimise cross-tree constraints (excludes and requires)

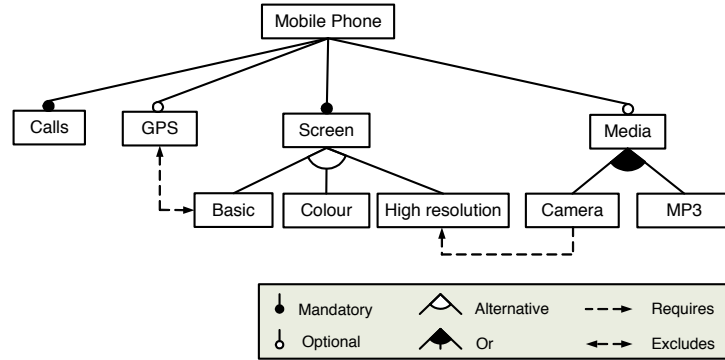
2. Develop Behavioural Model

- Select a formalism — Featured transition systems, Feature Petri nets
- Develop isolated behavioural models for each feature
- Combine isolated models into single model

3. Develop Implementation

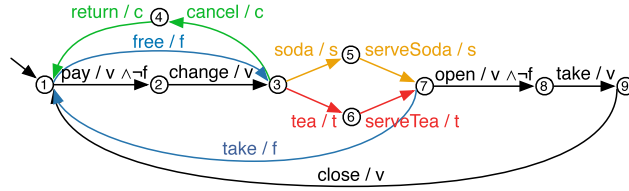
- Select approach — #ifdef vs delta-oriented
- Understand existing code modules
- Use selected approach to construct a product line from these modules. Begin incrementally, integrating a feature at a time.

3 Cheat Sheet: Feature Models



$$\begin{aligned}
 \text{root}(f) &\equiv f \\
 \text{mandatory}(p, f) &\equiv f \Leftrightarrow p \\
 \text{optional}(p, f) &\equiv f \Rightarrow p \\
 \text{alternative}(p, \{f_1, \dots, f_n\}) &\equiv ((f_1 \vee \dots \vee f_n) \Leftrightarrow p) \wedge \\
 &\quad \bigwedge_{i < j} \neg(f_i \wedge f_j) \\
 \text{or}(p, \{f_1, \dots, f_n\}) &\equiv (f_1 \vee \dots \vee f_n) \Leftrightarrow p
 \end{aligned}$$

4 Cheat Sheet: Featured Transition Systems



Each transition has the following form:

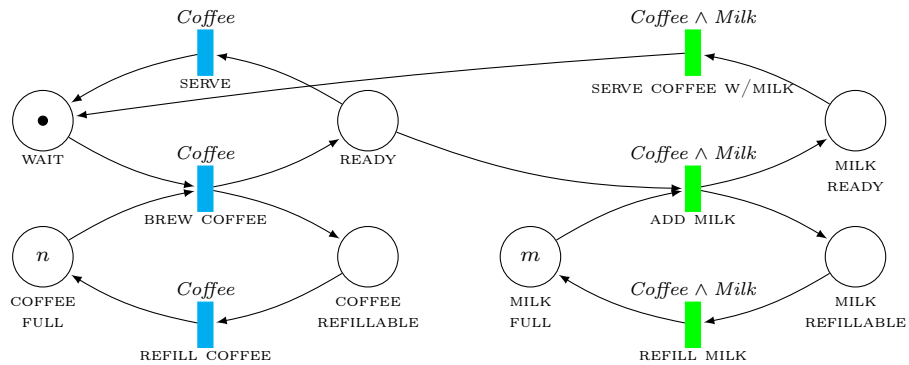
$$state \xrightarrow{action/application \ condition} state'$$

where an *application condition* is a predicate feature names.

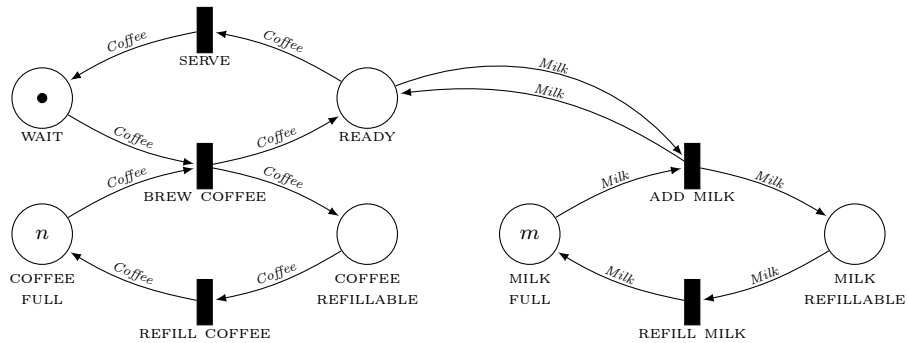
5 Cheat Sheet: Feature Petri Nets

Two different feature Petri net models are considered, transition-labelled and arc-labelled variants. In these models, application conditions are written above the transition and arc, respectively.

Transition-labelled Feature Petri Net:



Arc-labelled Feature Petri Net:



6 Cheat Sheet: #ifdef

Don't stick too strictly to the constraints imposed by the C preprocessor. Typically guards will be application conditions.

The following usage ensures that code `controlled text` appears in the final product if `GUARD` is true, based on statically available information.

```
#ifdef GUARD
    controlled text
#endif
```

The following has the expected meaning.

```

#ifdef GUARD
    controlled text
#else
    alternative text
#endif

```

You can also use `#ifndef` to test whether guard is false.

7 Cheat Sheet: Delta Modelling

Assume that the base language is Java or anything you are comfortable with.

Deltas

Deltas are modules that sit beside a class hierarchy and, when applied, can perform the following modifications of the hierarchy:

- Modifications on Class Level:
 - Addition, Removal and Modification of Classes.
- Modifications of Class Structure:
 - Changing interfaces
 - Adding/Removing Fields/Methods
 - Modifying Methods (wrapping original call)

The following delta modifies a method in a class, using `original` to call the original method.

```

delta Rpt (Int times) {
  modifies Greeter {
    modifies String say_hello() {
      String result = "";
      Int i = 0;
      while (i < times) {
        result = result + original();
        i = i + 1;
      }
      return result;
    }
  }
}

```

The following delta adds a method to the `Greeter` class:

```

delta Goodbye {
  modifies Greeter {
    add String say_goodbye() {

```

```

        return "Goodbye";
    }
}

```

Similarly, classes can be added to systems using deltas.

The following delta removes class `Greeter`.

```

delta Remove {
    removes Greeter
}

```

The following delta removes method `say_hello()` from class `Greeter`.

```

delta Remove {
    modifies Greeter {
        remove String say_hello();
    }
}

```

Product Line Configuration

```

product line HelloMultiLingual {
    features Repeat, German, French, Dutch, Swedish;
    core English;

    delta De when German && not Repeat;
    delta Fr when French;
    delta Nl when Dutch;
    delta Sv when Swedish && Repeat;
    delta Rpt(Repeat.times)
        after De, Fr, Nl, Sv when Repeat;
}

```

- `product line` names the product line.
- `feature` describes the features referred to in the product line.
- `core` identifies the core to which the features will be applied.
- `delta` refers to a delta and states **when** the delta is applicable using a so-called *application condition*.
- The parameter to `Rpt` comes from an attribute in the feature model associated with feature `Repeat`.
- `after` declares that delta `Rpt` must be applied after deltas `De`, `Fr`, `Nl`, `Sv`, if present.