Feature Modularity

Jo Atlee • FOSD • September 2014

WATFORM

David R. Cheriton School of Computer Science
University of Waterloo
feature-oriented software development

**feature**: a unit of *functionality* or *added value* in the product

- Call Forwarding
- Basic Call Service
- Billing

stakeholders’ mental model of system

feature-oriented software system
feature interactions

feature interaction: a feature behaves differently in the presence of another feature than it behaves in isolation

› unimplementable
› nondeterministic
› conflicting changes to shared context
› violates correctness property

anti-theft system
locks doors and windows
sounds alarm if vehicle is touched

accident response system
deploys airbags
deactivates fuel pump
disconnects battery
unlocks door
calls emergency personnel
not all interactions are bad!

**intended interactions**
- advanced cruise-control variants *override* basic cruise control
- prohibit navigation *overrides* navigation
- prohibit-navigation override *overrides* prohibit-navigation

---

**unintended but harmless interactions**
- call screening *prevents activation of* caller id

---

**(planned) resolutions to conflicts**
- brake override *overrides* (acceleration ⊕ braking)
fixing undesired interactions

- fix faulty feature
- disallow feature combination
- resolve interaction using exceptions
- resolve interaction through a new feature
feature interaction problem

- the number of potential interactions is exponential in the number of features

- death by exceptions [Zave]

  \[ F_1 = f_1 + e_{f_2} + e_{f_3} + ... + e_{f_n} \]

- feature development is dominated by tasks related to addressing interactions
lots of features

e.g., telephony has 1000+ features per system

a system of feature-rich systems

› features from multiple providers
› multiple active versions of the same feature
lots of interactions

results of the second feature interaction contest
lots of types of interactions

**control-flow**
one feature affects the flow of control in another feature

**data-flow**
one feature affects (deletes, alters) a message destined for another feature

**data modification**
shared data read by one feature is modified by another feature

**data conflict**
two features modify the same data

**control conflicts**
two features issue conflicting actions

**assertion violation**
one feature violates another feature's assertions or invariants

**resource contention**
the supply of resources is inadequate, given the set of competing features
feature-orientation vs. interactions

FOSD emphasizes features, de-emphasizes interactions

• annotative approach
  – interactions manifest as nested preprocessor directives
  – which state how all features interact

• compositional approach
  – interactions realized (implicitly) by composition
  – fixes realized through new “feature” modules

this is exactly the chore that feature-orientation was meant to avoid!
take aways

1. resolve interactions *en masse* outside of features
feature modules (no interfaces)
feature modules

features are modelled as hierarchical state machines
additive evolution

a new feature may...

- introduce behaviours
  - via: new machines

- eliminate behaviours
  - via: new or stronger enabling conditions on existing actions or transitions

- substitute behaviours
  - via: new *pre-empting* actions or transitions

intended interactions:
modelled as structural extensions (fragments)
at extension points in existing features
extension points

state machine main

let car = myproduct.Controls.car

extension points

state (new region, new sub machine)

region (new states, new transitions)

transition (new guard, new action)

guard (new clause)
feature modules

features can be modelled as extensions to existing features

Cruise Control (CC)
feature modules

features can be modelled as extensions to existing features

extension point

new guard clause

state-machine extension

BDS\{t3}\: \textbf{[strengthen with } c : \textbf{not inState(main.enabled.main.engaged.main.active)} \textbf{or driverOverride()}\]

Cruise Control (CC)
explicate interactions

intended interactions, overrides, priorities should be explicit

Headway Control (HC)
explicate interactions

intended interactions, overrides, priorities should be explicit

Headway Control (HC)
take aways

1. resolve interactions *en masse outside of features*

2. **feature modularity** to ease complexity, promote parallel development
   - express fragments wrt extension points
   - explicate intended interactions
feature interfaces
interfaces and information hiding

**interface** advertises what services a module provides to the rest of the system, and how they can be accessed.

**information hiding** encapsulates a design decision inside a module, whose interface reveals only externally visible properties [Parnas’72]
feature interface would define what services a feature provides to the rest of the system and how other features can access those services.
generic feature interface

most inter-feature references are to high-level common modes of operation
Text: **[FeatureX_Fail]** flag shall be set to true when **Feature Y** is in **fail** state…

Feature X

Feature Y
generic feature interface (2)
take aways

1. resolve interactions en masse outside of features

2. feature modularity to ease complexity, promote parallel development
   - express fragments wrt extension points
   - explicate intended interactions

3. (public) feature interfaces hide implementation details
   - expose feature’s inputs/outputs, accessors
   - generic interface exposes behaviour modes
feature composition
feature structure trees (FSTs)

non-terminal node

terminal node

BDS FST
compose feature modules by **superimposing** their feature structure trees (FSTs)

composition is **commutative** and **associative** because terminal nodes are not merged
superimposition

partial BDS FST
superimposition

partial CC FST
superimposition

partial HC FST
resulting composition (product)

composition is a collection of parallel machines that have been extended with fragments
commutativity

**non-commutative:**
Intended interactions realized by *implicit total order*  
(e.g., DFC, AHEAD)

- Resolves unknown conflicts
- Undesired resolutions
- Analyze multiple orderings
- Recompute order for new feature
- Implicit intended interactions

**commutative:**
Intended interactions specified by *explicit partial order*  
(e.g., transition and action priorities)

- Explicit intended interactions
- Only specify desired resolutions
- Analyze single feature order
- Ease of adding new feature
- Detect unknown conflicts
take aways

1. resolve interactions *en masse outside of features*

2. **feature modularity** to ease complexity, promote parallel development
   - express fragments wrt extension points
   - explicate intended interactions

3. **feature (public) interfaces** hide implementation details
   - expose feature’s inputs/outputs, accessors, mutators
   - generic interface exposes behaviour modes

4. **commutative composition**
resolving unintended interactions
feature coordination

- fixed set of features
- pre-determined selection of features
- static integration
- perfect coordination possible

- fixed set of features
- semi-configurable selection of features
- set of static integrations
- perfect coordination possible, but impractical

- unlimited features
- user-defined selection of features
- dynamic integration
- loose coordination
feature coordination

composition is a collection of parallel machines that have been extended with fragments

each machine’s interface is simply its inputs and outputs
serializing features
Distributed Feature Composition [Jackson, Zave, TSE’98]

pipeline architecture

+ features make no assumptions about other features
+ avoids simultaneous reactions to the same event
+ conflicts are resolved through serialization
+ feature ordering realizes a priority scheme
− resolution is implicit
parallel execution (resolution modules)

+ features make no assumptions about other features
+ conflicting actions are resolved by resolution modules
+ all actions are considered in resolution
+ resolution strategies can be variable-specific
**summary**

**modular features**
- extension points
- intended interactions

**generic public interfaces**
- mode of operation

**composition**
- of feature families
- commutative

**coordination**
- of compositions
- relax “correctness”
- focus on safety

Resolve \( r O_1 \)

Resolve \( r O_n \)