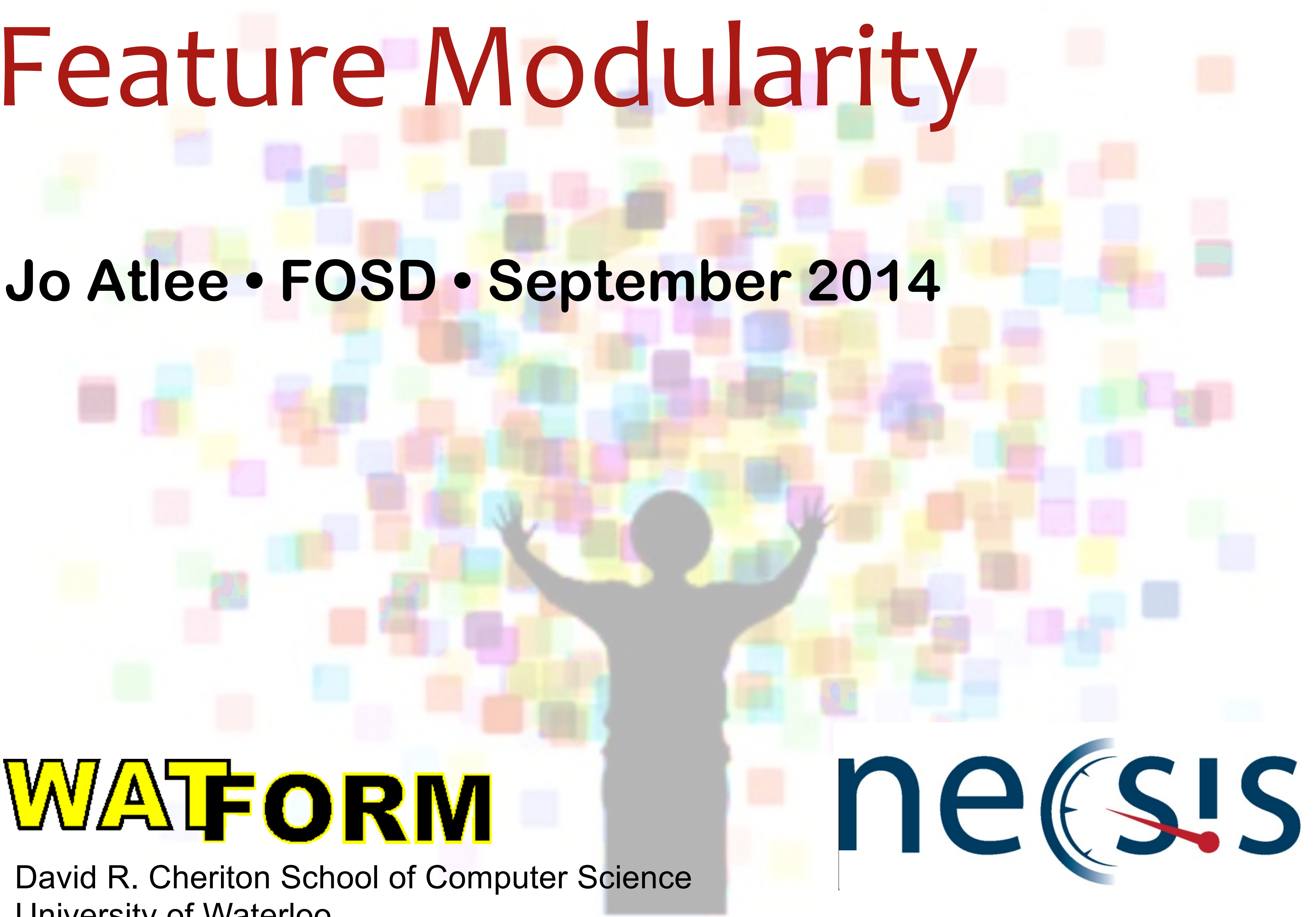
Feature Modularity

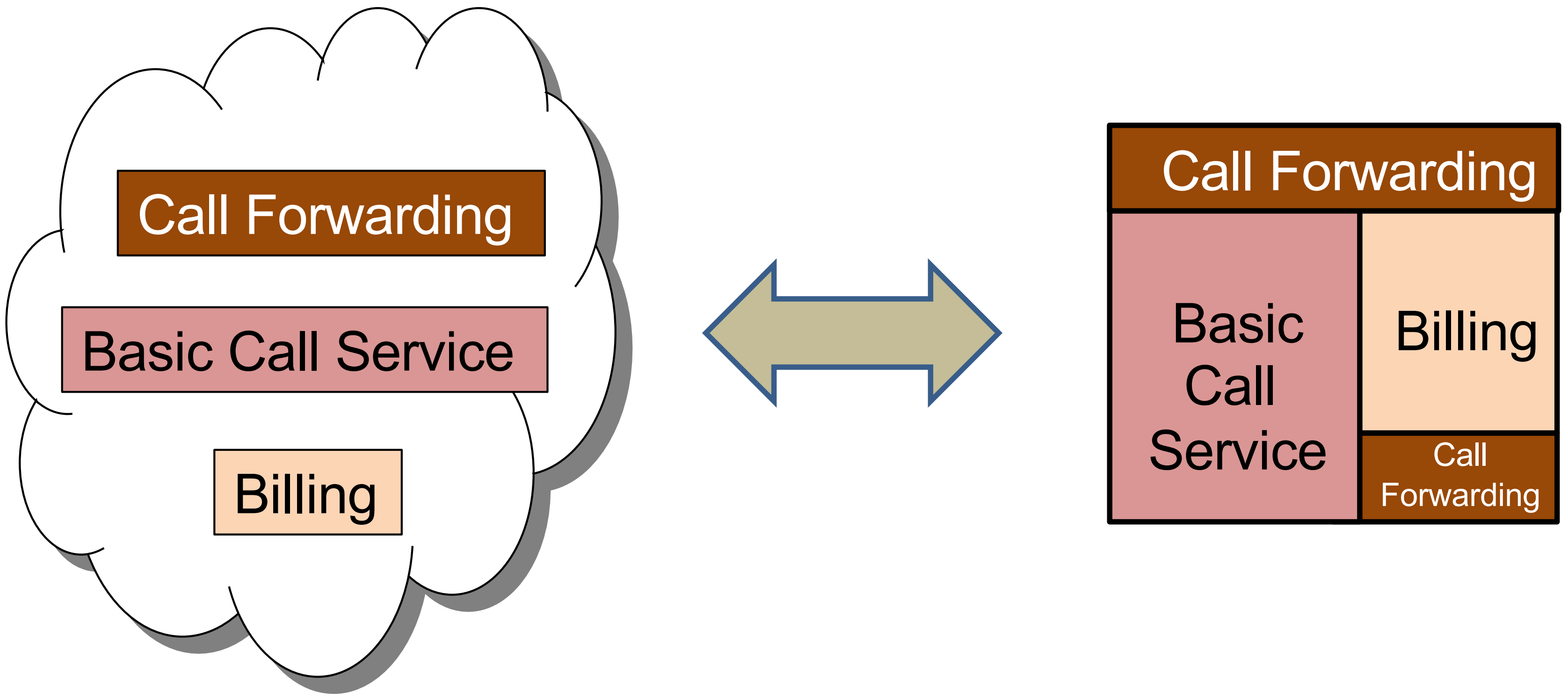
Jo Atlee • FOSD • September 2014



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



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

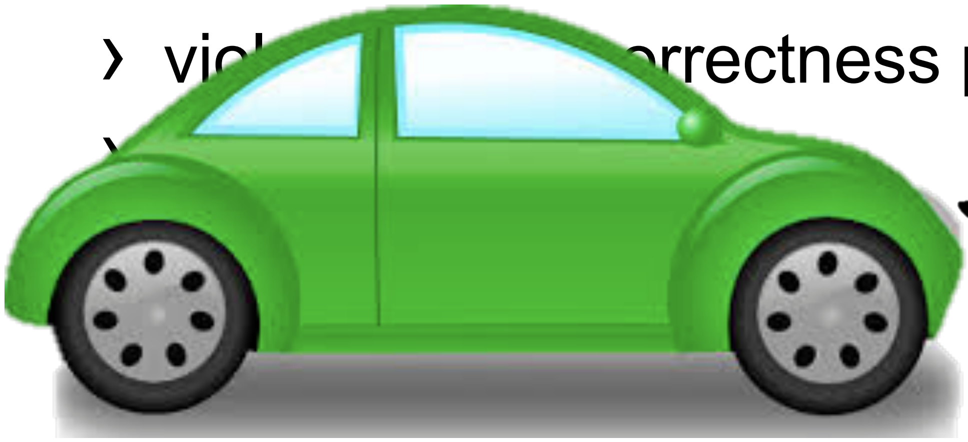
- unimplementable
- nondeterministic
- conflicting changes to shared context
- vi [illegible] rrectness property
- [illegible]

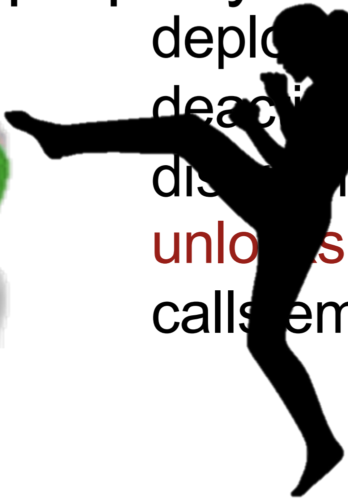
anti-theft system

locks doors and windows
sounds alarm if vehicle is touched

accident response system

deplo [illegible] airbags
deac [illegible] tes fuel pump
di [illegible] ects battery
unlo [illegible] s 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

---

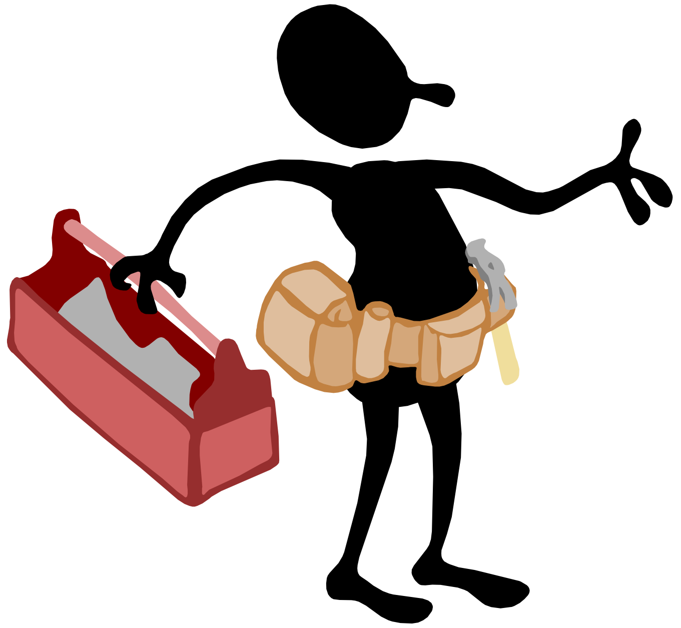
unintended but harmless interactions

- call screening prevents activation of caller id

(planned) resolutions to conflicts

- brake override overrides (acceleration $\oplus$ braking)

fixing undesired interactions



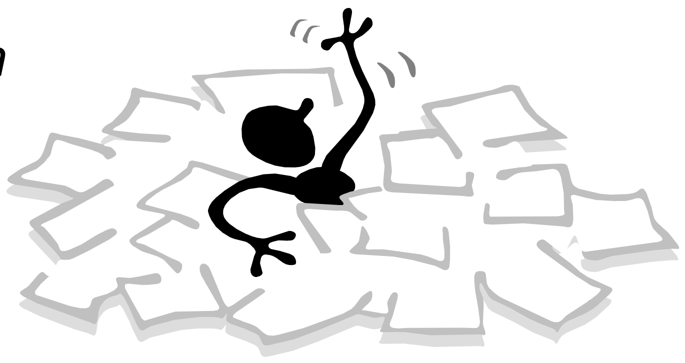
- fix faulty feature
- disallow feature combination
- resolve interaction using exceptions
- resolve interaction through a new feature

feature interaction problem

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

- death by exceptions [Zave]

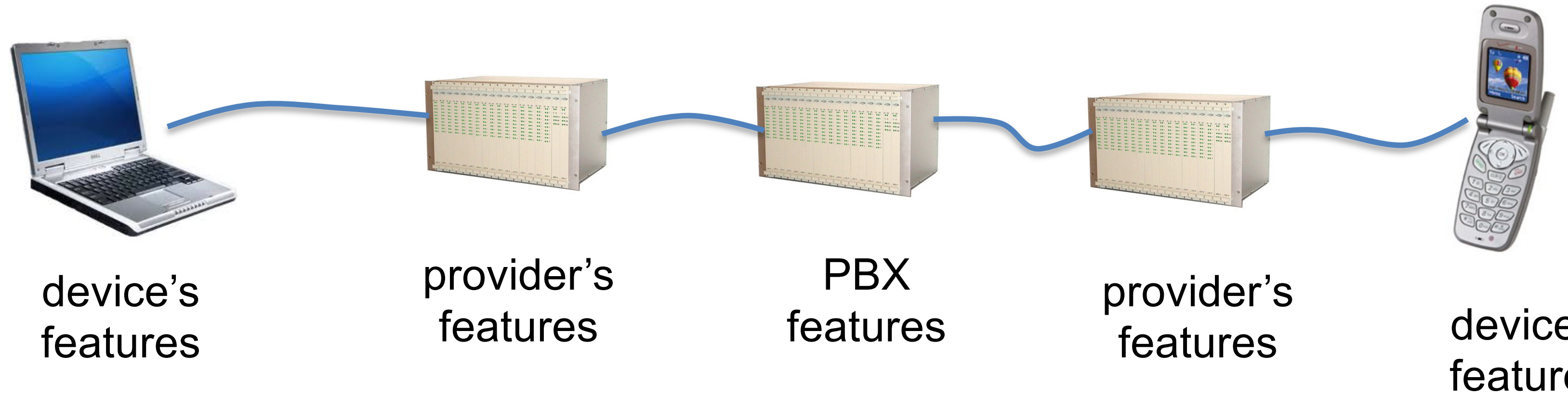
$$F_1 = f_1 + e_{f_2} + e_{f_3} + \ldots + e_{f_n}$$



- feature development is dominated by tasks related to addressing interactions

lots of features

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

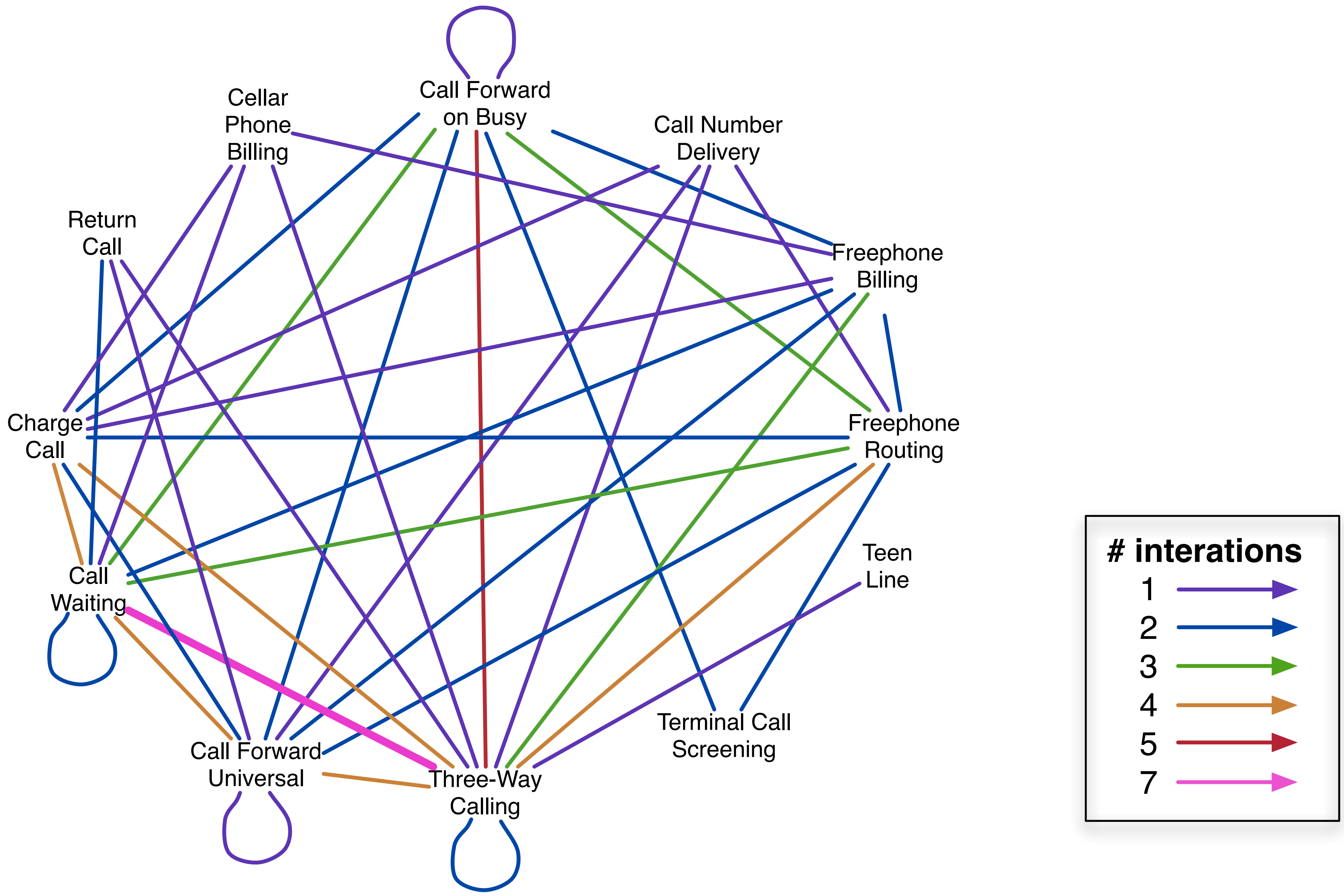


a system of feature-rich systems

- 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

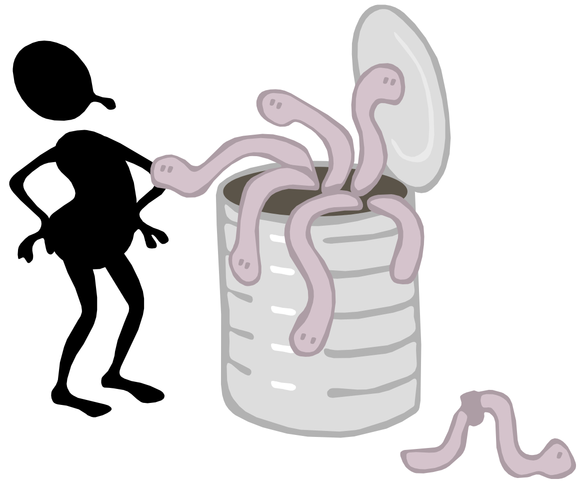
resource contention
the supply of resources is inadequate, given the set of competing features

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

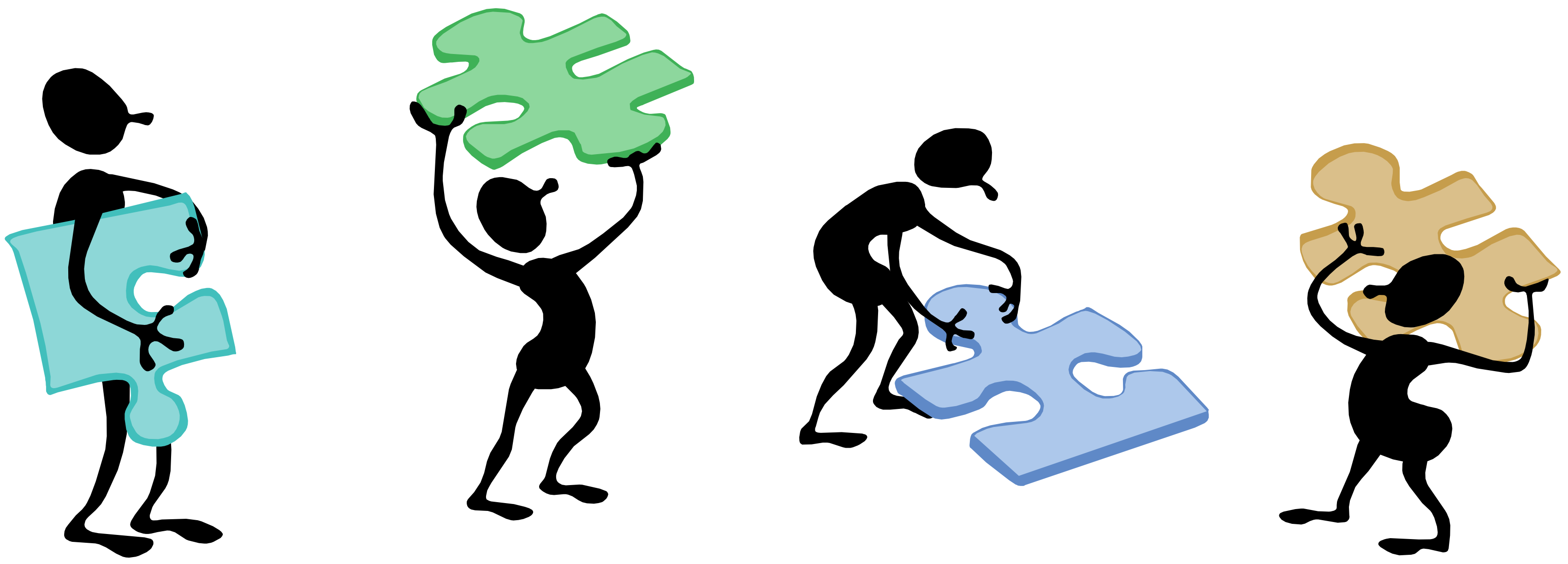
this is exactly the chore that feature-orientation was meant to avoid!



take aways

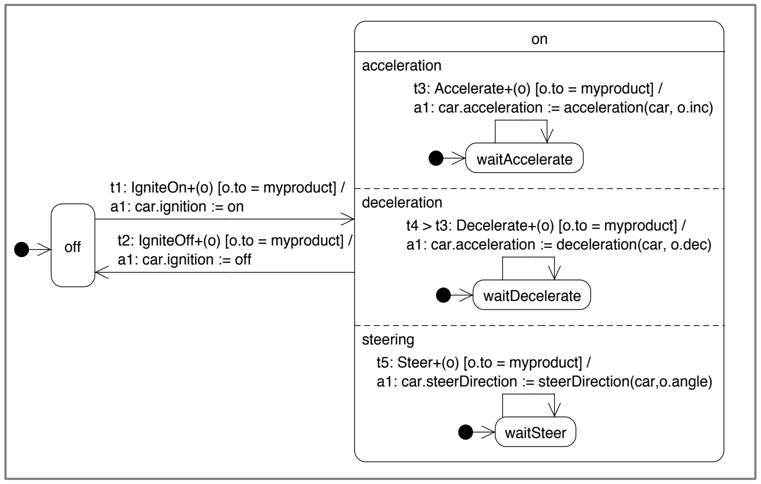
1. **resolve interactions en masse outside of features**

feature modules (no interfaces)

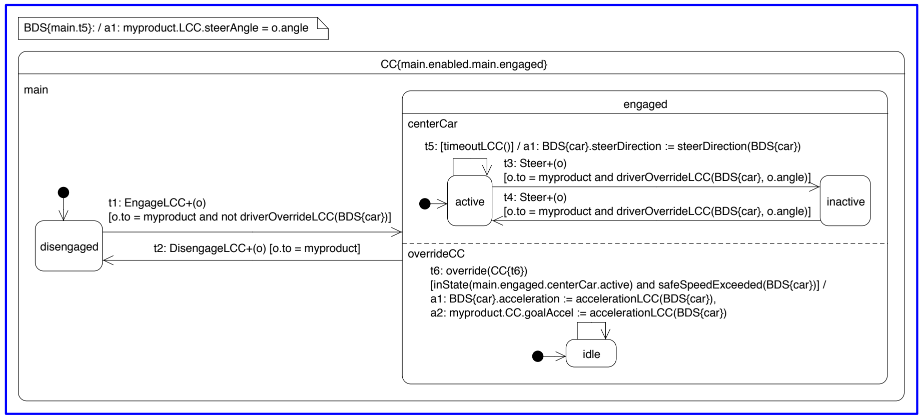


feature modules

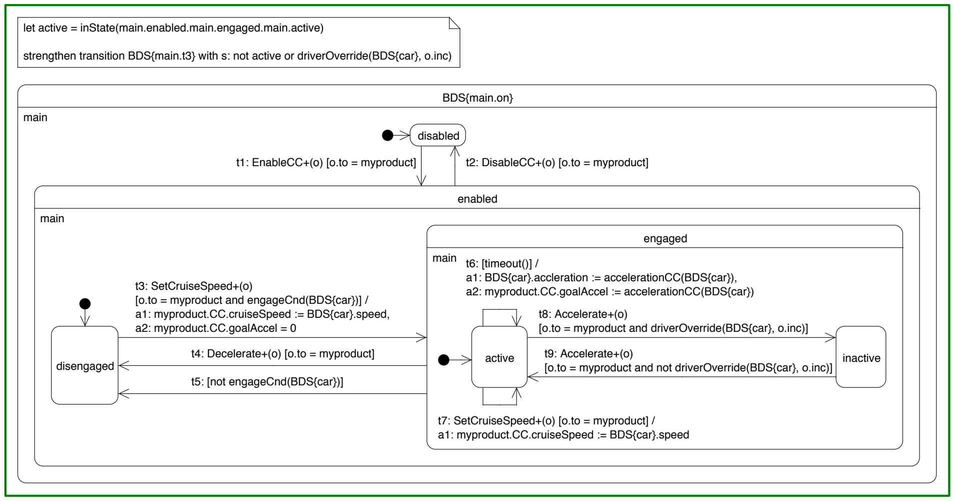
features are modelled as hierarchical state machines



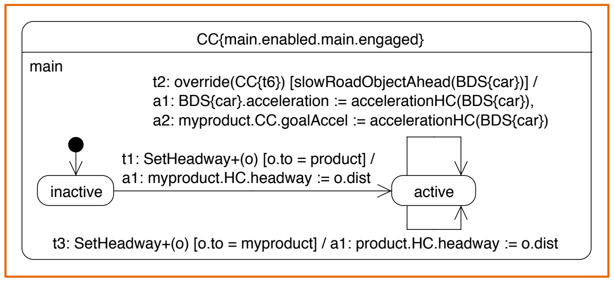
on
acceleration
t3: Accelerate+(o) [o.to = myproduct] /
a1: car.acceleration := acceleration(car, o.inc)
waitAccelerate
deceleration
t4 > t3: Decelerate+(o) [o.to = myproduct] /
a1: car.acceleration := deceleration(car, o.dec)
waitDecelerate
steering
t5: Steer+(o) [o.to = myproduct] /
a1: car.steerDirection := steerDirection(car,o.angle)
waitSteer
t1: IgniteOn+(o) [o.to = myproduct] /
a1: car.ignition := on
t2: IgniteOff+(o) [o.to = myproduct] /
a1: car.ignition := off
off



BDS{main.t5}: / a1: myproduct.LCC.steerAngle = o.angle
CC{main.enabled.main.engaged}
main
engaged
centerCar
t5: [timeoutLCC()] / a1: BDS{car}.steerDirection := steerDirection(BDS{car})
t3: Steer+(o)
[o.to = myproduct and driverOverrideLCC(BDS{car}, o.angle)]
active
t4: Steer+(o)
[o.to = myproduct and driverOverrideLCC(BDS{car}, o.angle)]
inactive
t1: EngageLCC+(o)
[o.to = myproduct and not driverOverrideLCC(BDS{car})]
disengaged
t2: DisengageLCC+(o) [o.to = myproduct]
overrideCC
t6: override(CC{t6})
[inState(main.engaged.centerCar.active) and safeSpeedExceeded(BDS{car})] /
a1: BDS{car}.acceleration := accelerationLCC(BDS{car}),
a2: myproduct.CC.goalAccel := accelerationLCC(BDS{car})
idle



let active = inState(main.enabled.main.engaged.main.active)
strengthen transition BDS{main.t3} with s: not active or driverOverride(BDS{car}, o.inc)
BDS{main.on}
main
disabled
t1: EnableCC+(o) [o.to = myproduct]
t2: DisableCC+(o) [o.to = myproduct]
enabled
main
engaged
main
t6: [timeout()] /
a1: BDS{car}.accleration := accelerationCC(BDS{car}),
a2: myproduct.CC.goalAccel := accelerationCC(BDS{car})
t3: SetCruiseSpeed+(o)
[o.to = myproduct and engageCnd(BDS{car})] /
a1: myproduct.CC.cruiseSpeed := BDS{car}.speed,
a2: myproduct.CC.goalAccel = 0
t8: Accelerate+(o)
[o.to = myproduct and driverOverride(BDS{car}, o.inc)]
t4: Decelerate+(o) [o.to = myproduct]
active
t9: Accelerate+(o)
[o.to = myproduct and not driverOverride(BDS{car}, o.inc)]
inactive
disengaged
t5: [not engageCnd(BDS{car})]
t7: SetCruiseSpeed+(o) [o.to = myproduct] /
a1: myproduct.CC.cruiseSpeed := BDS{car}.speed



CC{main.enabled.main.engaged}
main
t2: override(CC{t6}) [slowRoadObjectAhead(BDS{car})] /
a1: BDS{car}.acceleration := accelerationHC(BDS{car}),
a2: myproduct.CC.goalAccel := accelerationHC(BDS{car})
t1: SetHeadway+(o) [o.to = product] /
a1: myproduct.HC.headway := o.dist
inactive
active
t3: SetHeadway+(o) [o.to = myproduct] / a1: product.HC.headway := o.dist

additive evolution

a new feature may…

- **introduce behaviours**
  - **via:** new machines

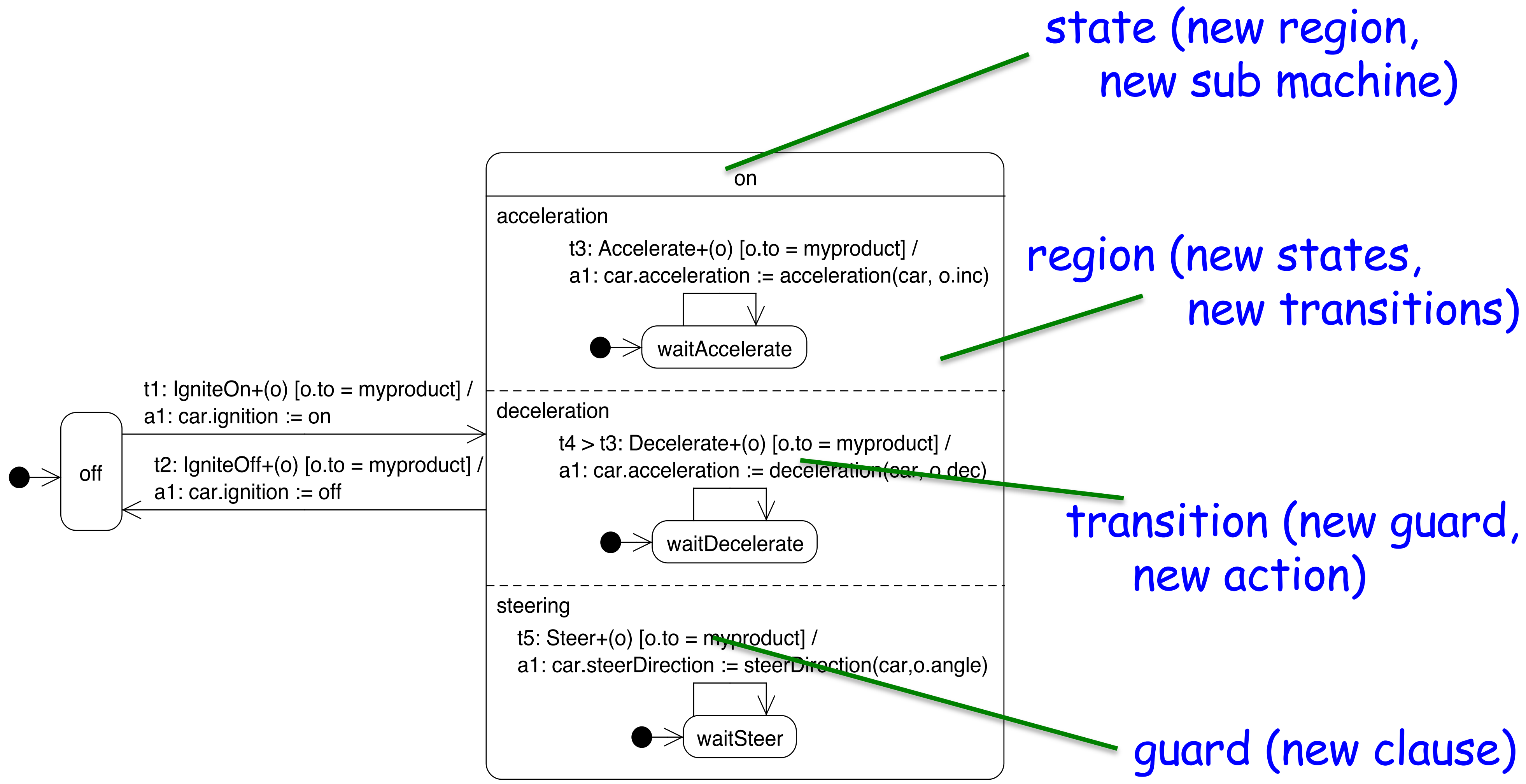
can also be expressed as fragments:
new regions, new states,
new transitions,
weakened enabling conditions

- **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

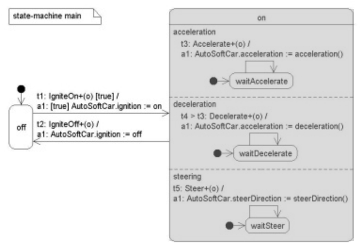
extension points


state (new region,
new sub machine)
region (new states,
new transitions)
transition (new guard,
new action)
guard (new clause)
on
acceleration
t3: Accelerate+(o) [o.to = myproduct] /
a1: car.acceleration := acceleration(car, o.inc)
waitAccelerate
deceleration
t4 > t3: Decelerate+(o) [o.to = myproduct] /
a1: car.acceleration := deceleration(car, o.dec)
waitDecelerate
steering
t5: Steer+(o) [o.to = myproduct] /
a1: car.steerDirection := steerDirection(car,o.angle)
waitSteer
off
t1: IgniteOn+(o) [o.to = myproduct] /
a1: car.ignition := on
t2: IgniteOff+(o) [o.to = myproduct] /
a1: car.ignition := off

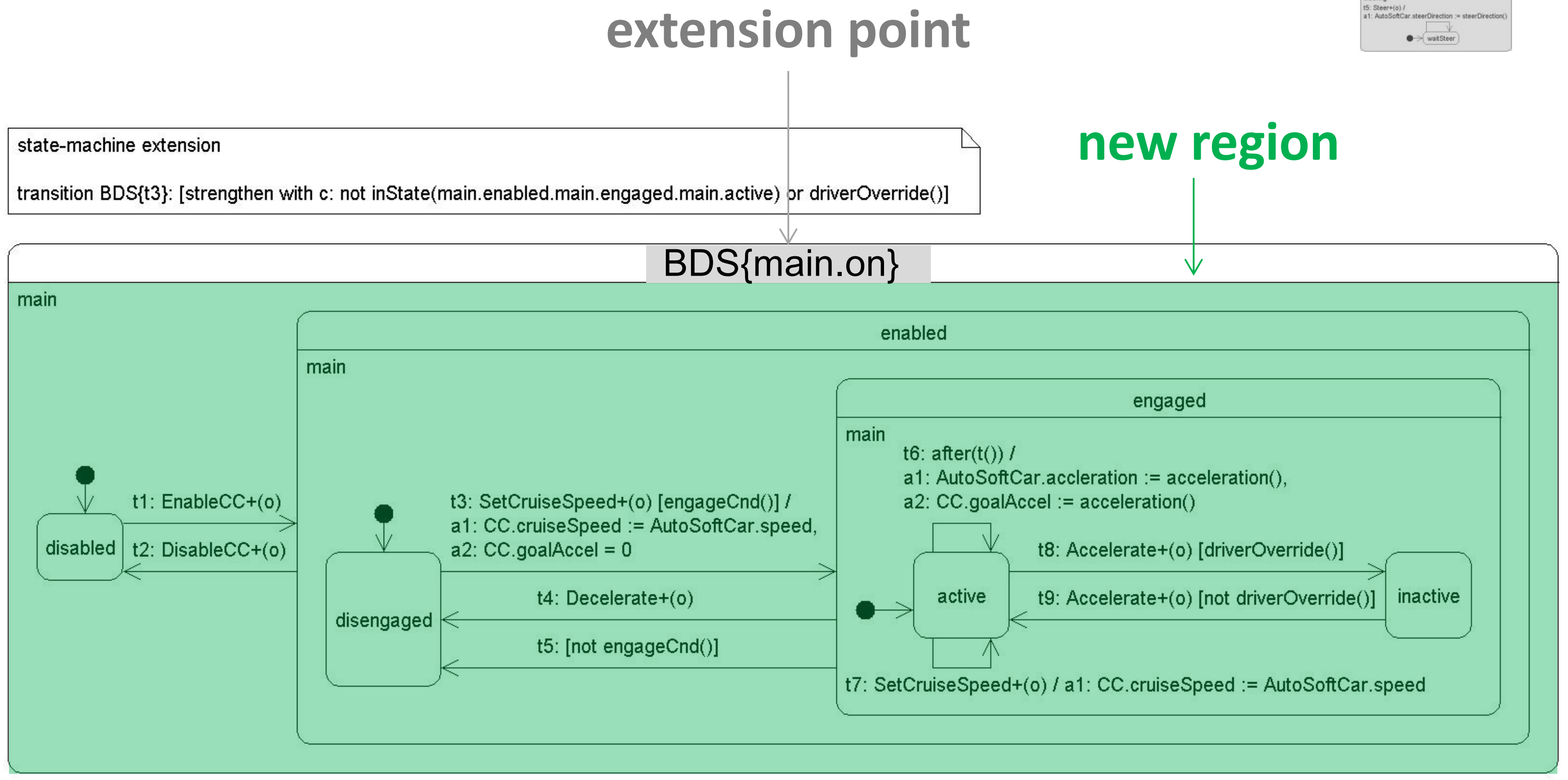
feature modules

features can be modelled as extensions to existing features

BDS



state-machine main
on
acceleration
t3: Accelerate+(o) /
a1: AutoSoftCar.acceleration := acceleration()
waitAccelerate
t1: IgniteOn+(o) [true] /
a1: [true] AutoSoftCar.ignition := on
off
t2: IgniteOff+(o) /
a1: AutoSoftCar.ignition := off
deceleration
t4 > t3: Decelerate+(o) /
a1: AutoSoftCar.acceleration := deceleration()
waitDecelerate
steering
t5: Steer+(o) /
a1: AutoSoftCar.steerDirection := steerDirection()
waitSteer



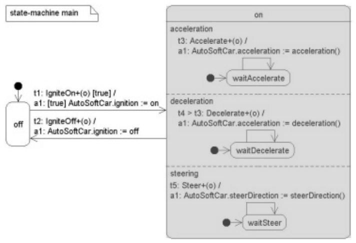
extension point
new region
state-machine extension
transition BDS{t3}: [strengthen with c: not inState(main.enabled.main.engaged.main.active) or driverOverride()]
BDS{main.on}
main
enabled
main
engaged
main
t6: after(t()) /
a1: AutoSoftCar.accleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: EnableCC+(o)
disabled
t2: DisableCC+(o)
t3: SetCruiseSpeed+(o) [engageCnd()] /
a1: CC.cruiseSpeed := AutoSoftCar.speed,
a2: CC.goalAccel = 0
active
t8: Accelerate+(o) [driverOverride()]
inactive
t4: Decelerate+(o)
t9: Accelerate+(o) [not driverOverride()]
disengaged
t5: [not engageCnd()]
t7: SetCruiseSpeed+(o) / a1: CC.cruiseSpeed := AutoSoftCar.speed

Cruise Control (CC)

feature modules

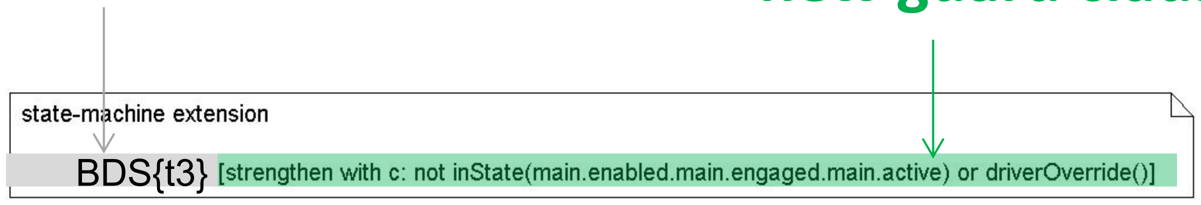
features can be modelled as extensions to existing features

BDS

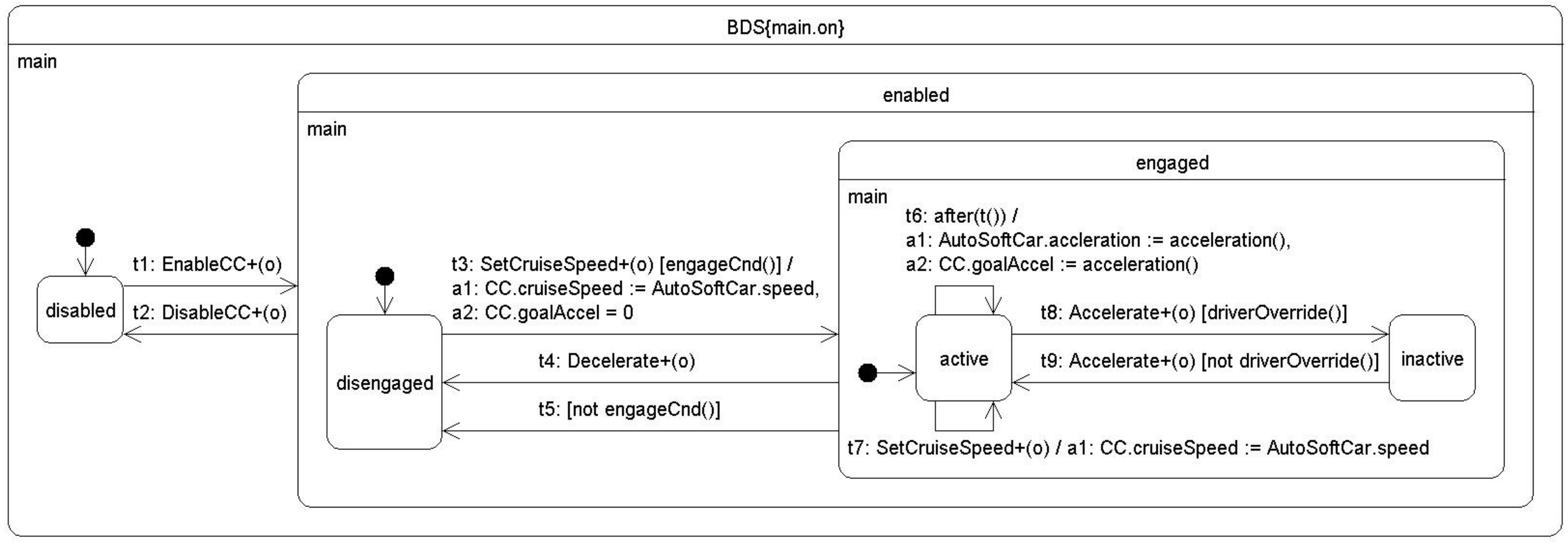

state-machine main
on
acceleration
t3: Accelerate+(o) /
a1: AutoSoftCar.acceleration := acceleration()
waitAccelerate
t1: IgniteOn+(o) [true] /
a1: [true] AutoSoftCar.ignition := on
off
t2: IgniteOff+(o) /
a1: AutoSoftCar.ignition := off
deceleration
t4 > t3: Decelerate+(o) /
a1: AutoSoftCar.acceleration := deceleration()
waitDecelerate
steering
t5: Steer+(o) /
a1: AutoSoftCar.steerDirection := steerDirection()
waitSteer

extension point

new guard clause


state-machine extension

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

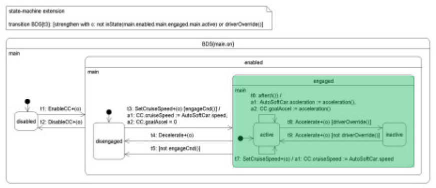

BDS{main.on}
main
enabled
main
engaged
main
t6: after(t()) /
a1: AutoSoftCar.accleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: EnableCC+(o)
disabled
t2: DisableCC+(o)
t3: SetCruiseSpeed+(o) [engageCnd()] /
a1: CC.cruiseSpeed := AutoSoftCar.speed,
a2: CC.goalAccel = 0
disengaged
t4: Decelerate+(o)
t5: [not engageCnd()]
active
t8: Accelerate+(o) [driverOverride()]
t9: Accelerate+(o) [not driverOverride()]
inactive
t7: SetCruiseSpeed+(o) / a1: CC.cruiseSpeed := AutoSoftCar.speed

Cruise Control (CC)

explicate interactions

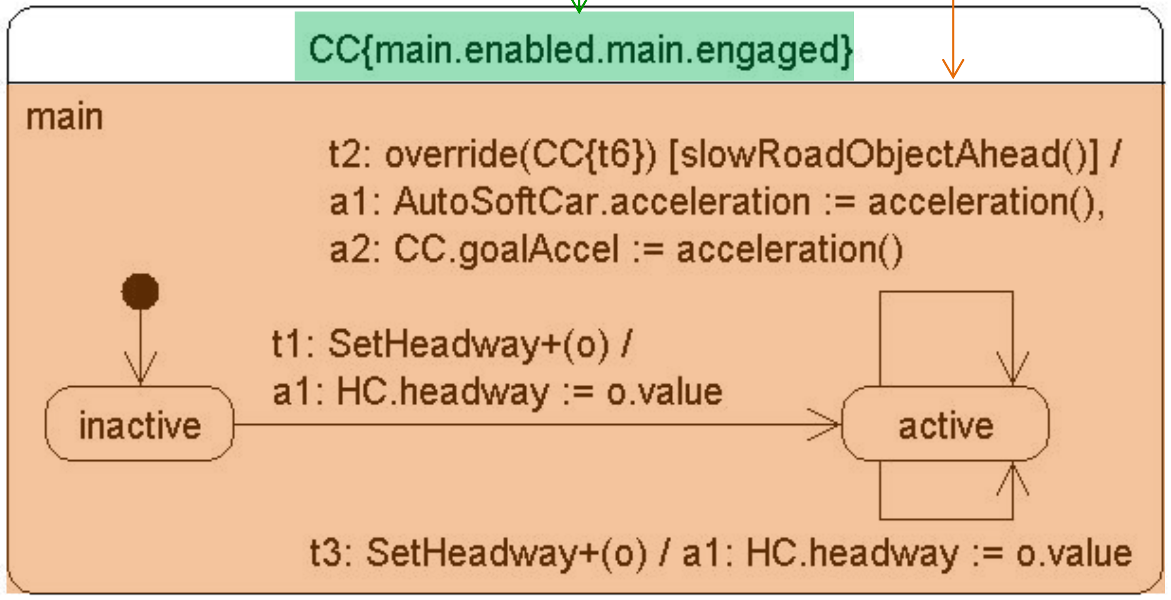
intended interactions, overrides, priorities should be explicit

CC



extension point

new region

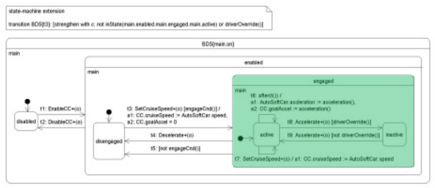

CC{main.enabled.main.engaged}
main
t2: override(CC{t6}) [slowRoadObjectAhead()] /
a1: AutoSoftCar.acceleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: SetHeadway+(o) /
a1: HC.headway := o.value
inactive
active
t3: SetHeadway+(o) / a1: HC.headway := o.value

Headway Control (HC)

explicate interactions

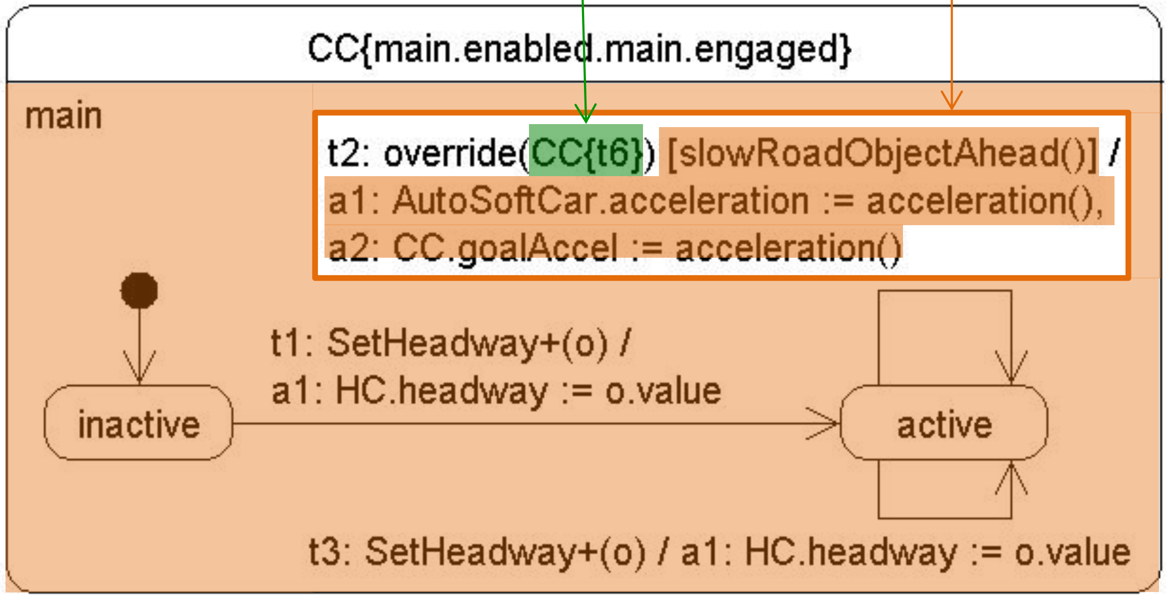
intended interactions, overrides, priorities should be explicit

CC


state-machine extension
transition BDS{t3}: [strengthen with c: not inState(main.enabled.main.engaged.main.active) or driverOverride()]
BDS{main.on}
main
enabled
main
engaged
main
t6: after(t()) /
a1: AutoSoftCar.acceleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: EnableCC+(o)
t2: DisableCC+(o)
disabled
t3: SetCruiseSpeed+(o) [engageCnd()] /
a1: CC.cruiseSpeed := AutoSoftCar.speed,
a2: CC.goalAccel = 0
disengaged
t4: Decelerate+(o)
t5: [not engageCnd()]
active
t8: Accelerate+(o) [driverOverride()]
t9: Accelerate+(o) [not driverOverride()]
inactive
t7: SetCruiseSpeed+(o) / a1: CC.cruiseSpeed := AutoSoftCar.speed

extension point

(new region includes pre-empting transition)


CC{main.enabled.main.engaged}
main
t2: override(CC{t6}) [slowRoadObjectAhead()] /
a1: AutoSoftCar.acceleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: SetHeadway+(o) /
a1: HC.headway := o.value
inactive
active
t3: SetHeadway+(o) / a1: HC.headway := o.value

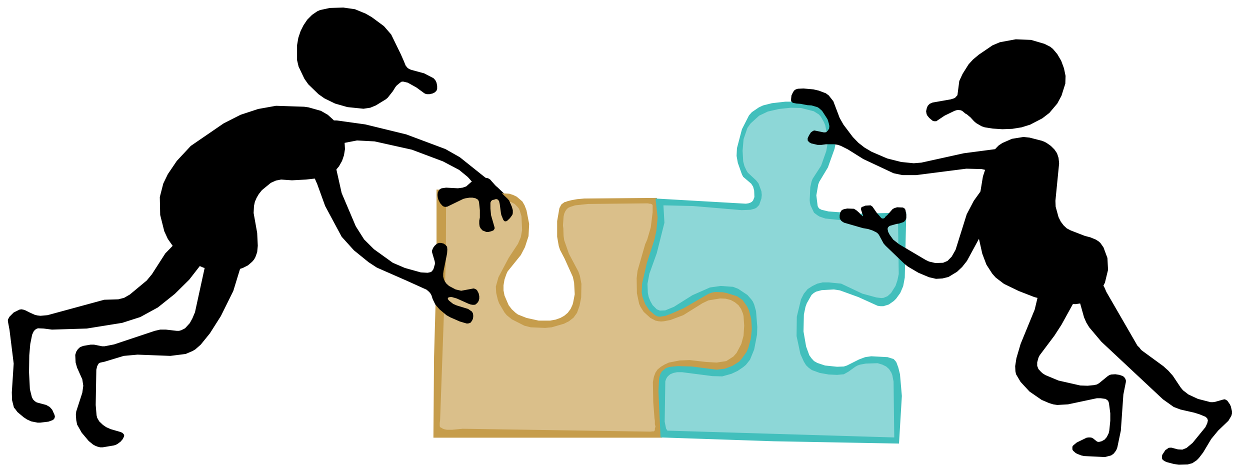
Headway Control (HC)

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

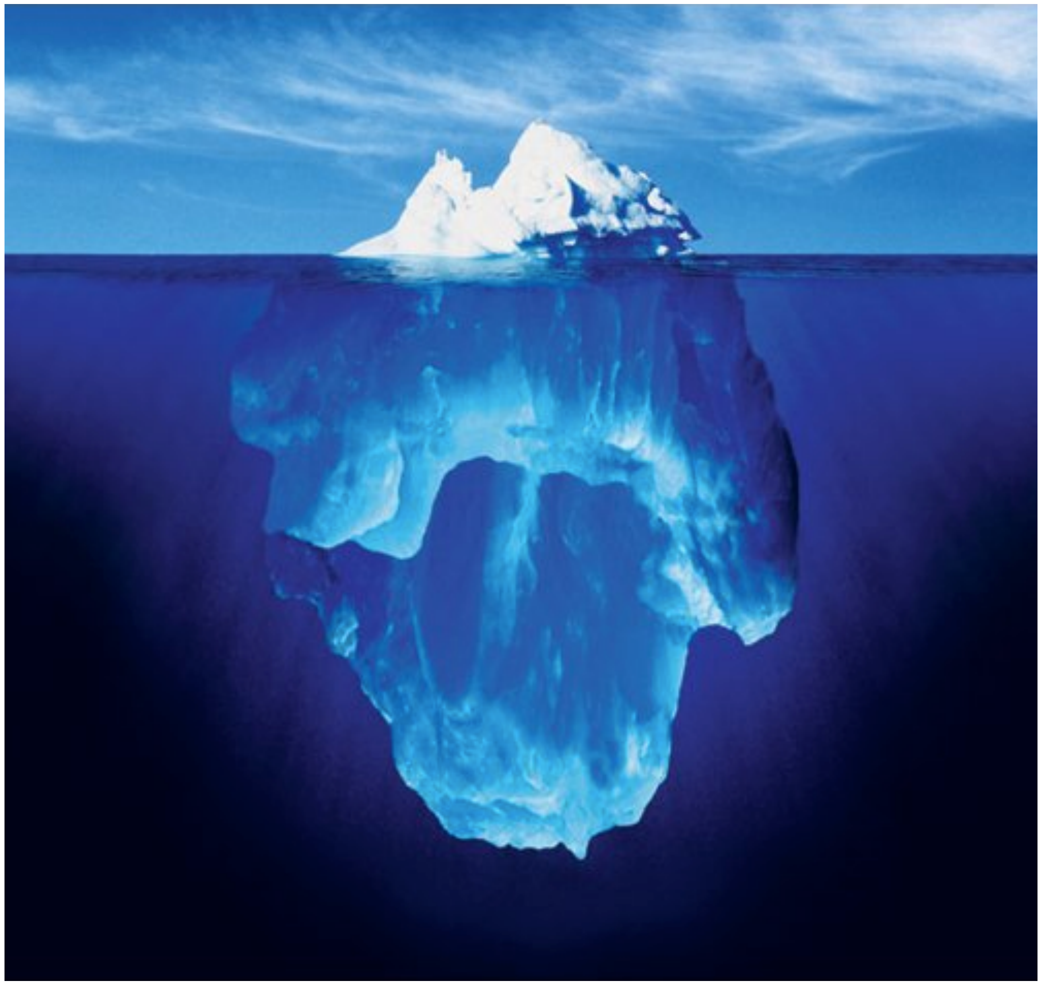
feature interfaces



interfaces and information hiding

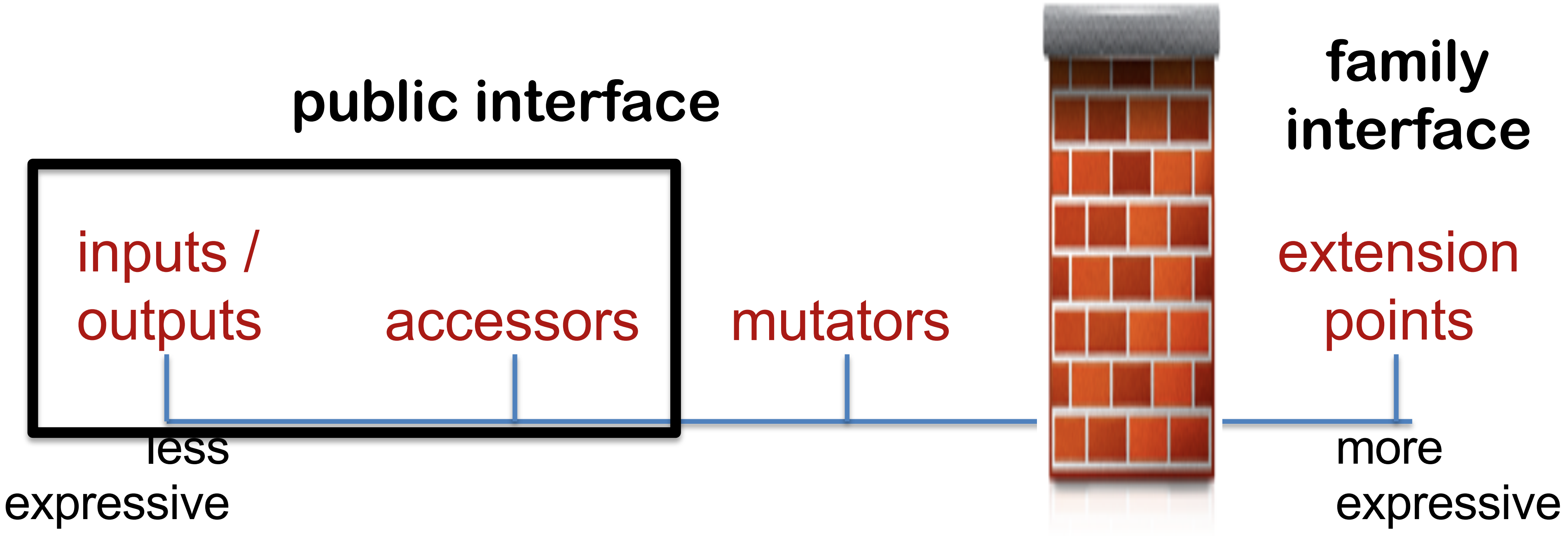
interface advertises what services a module provides to the rest of the system, and how they can be accessed

information hiding encapsulates a design decision inside a module, whose interface reveals only externally visible properties [Parnas'72]



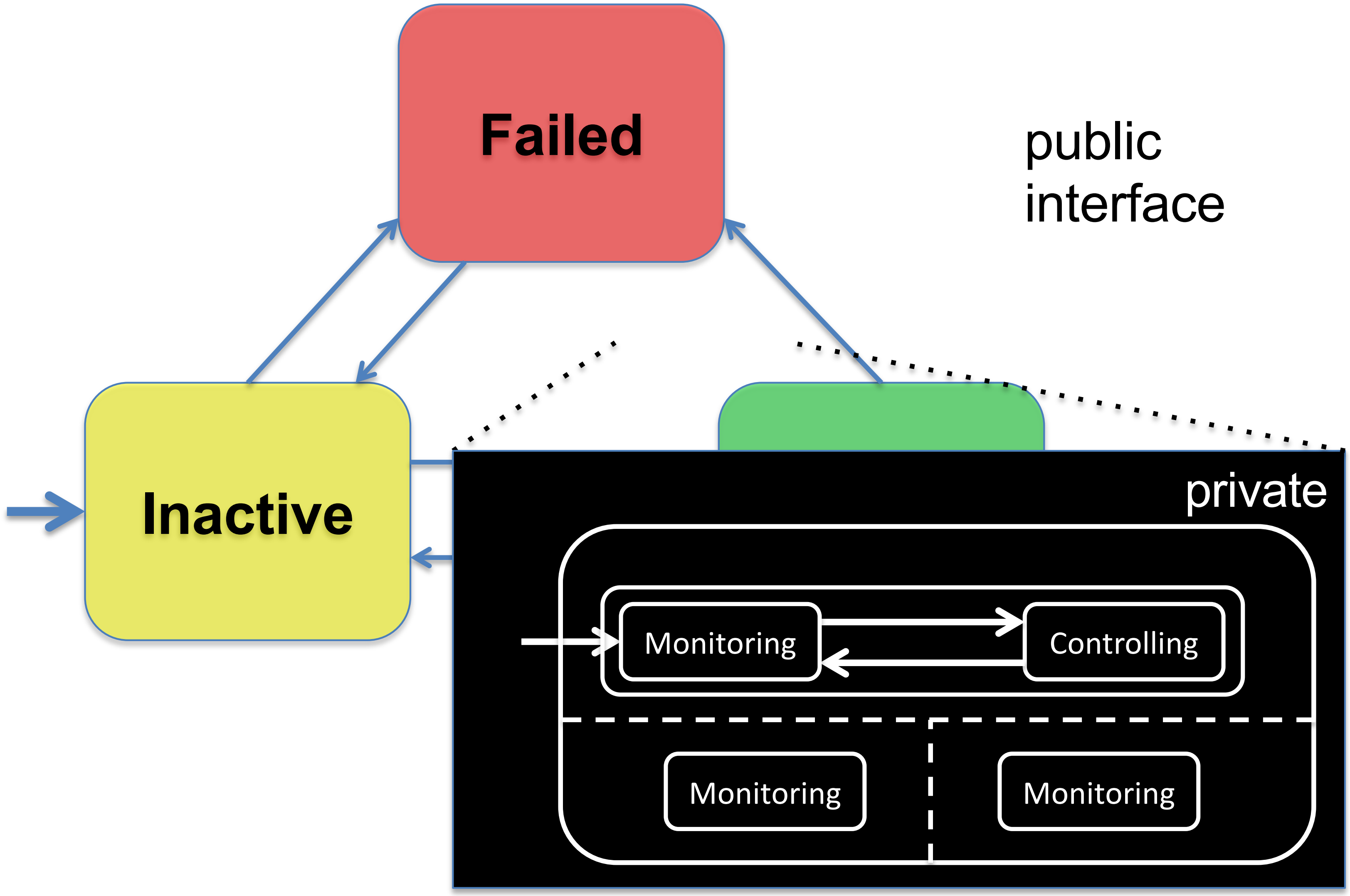
interfaces

feature interface would define what services a feature provides to the rest of the system and how other features can access those services



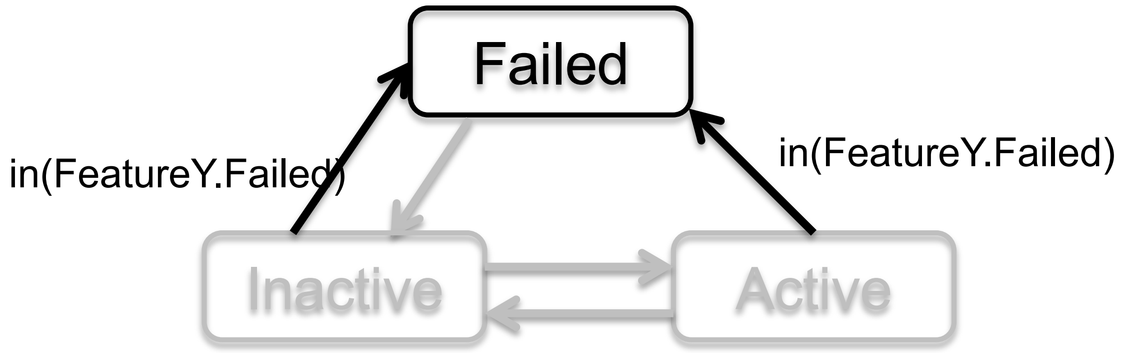
generic feature interface

most inter-feature references are to high-level common modes of operation

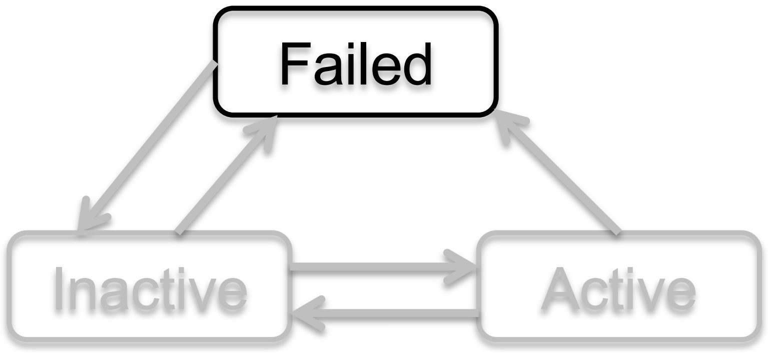


example

Text: *[FeatureX_Fail] flag shall be set to true when FeatureY is in fail state…*

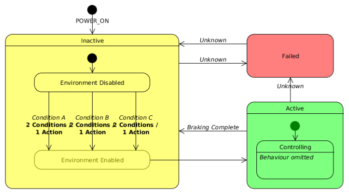


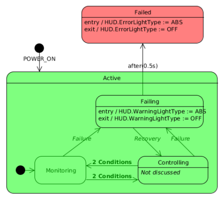
Feature X



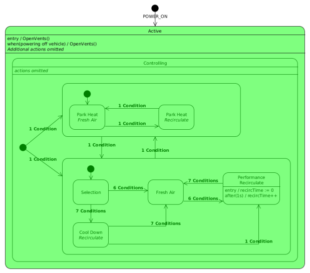
Feature Y

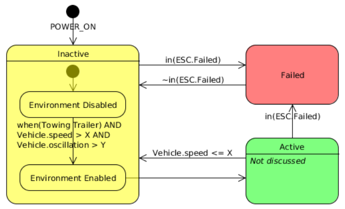
generic feature interface

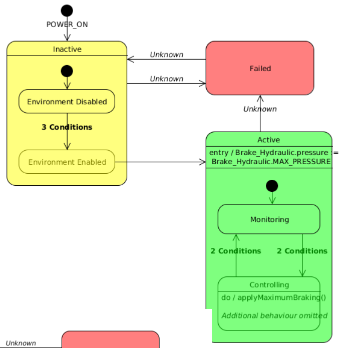


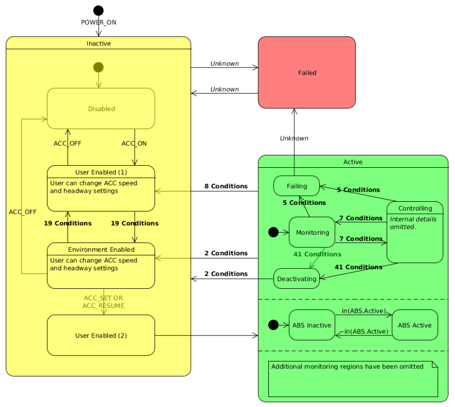


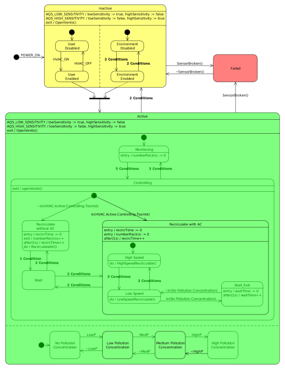
Failed
entry / HUD.ErrorLightType := ABS
exit / HUD.ErrorLightType := OFF
POWER_ON
after(0.5s)
Active
Failing
entry / HUD.WarningLightType := ABS
exit / HUD.WarningLightType := OFF
Failure
Recovery
Failure
Monitoring
2 Conditions
2 Conditions
Controlling
Not discussed



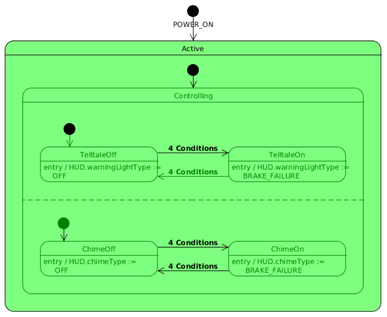




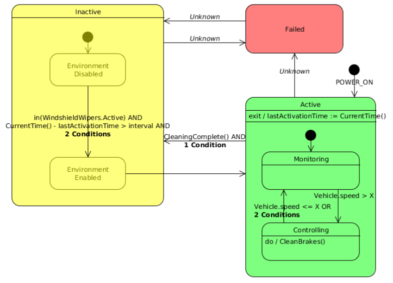


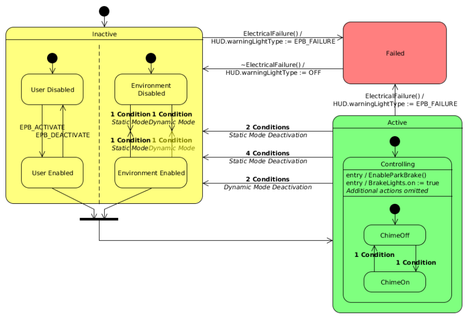


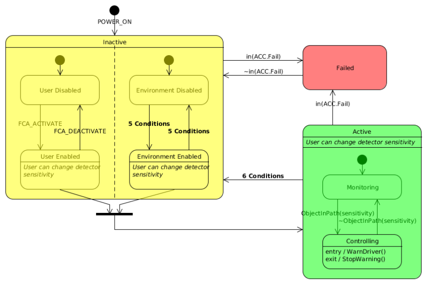
Inactive
POWER_ON
Failed
2 Conditions
Active
Monitoring
Controlling
Recirculate with AC
High Speed
Low Speed
No Pollution Concentration
Low Pollution Concentration
Medium Pollution Concentration
High Pollution Concentration

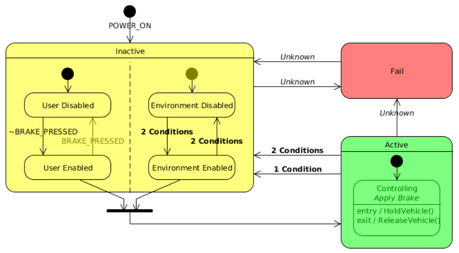


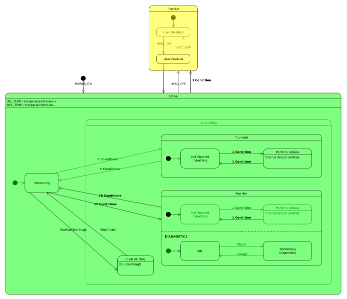
POWER_ON
Active
Controlling
TelltaleOff
entry / HUD.warningLightType := OFF
4 Conditions
4 Conditions
TelltaleOn
entry / HUD.warningLightType := BRAKE_FAILURE
ChimeOff
entry / HUD.chimeType := OFF
4 Conditions
4 Conditions
ChimeOn
entry / HUD.chimeType := BRAKE_FAILURE



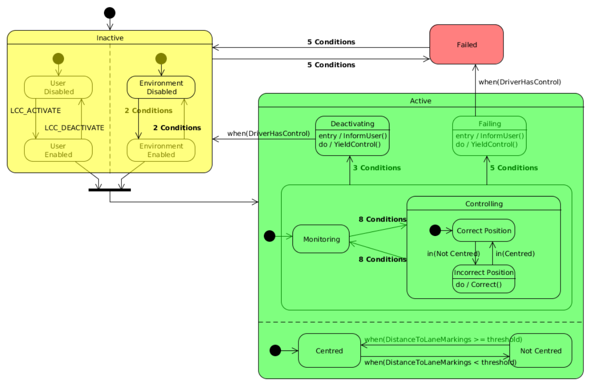


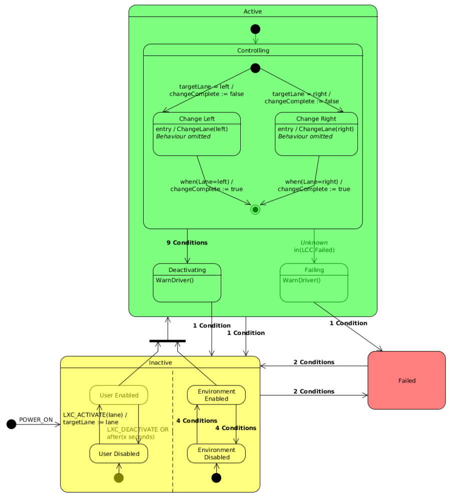


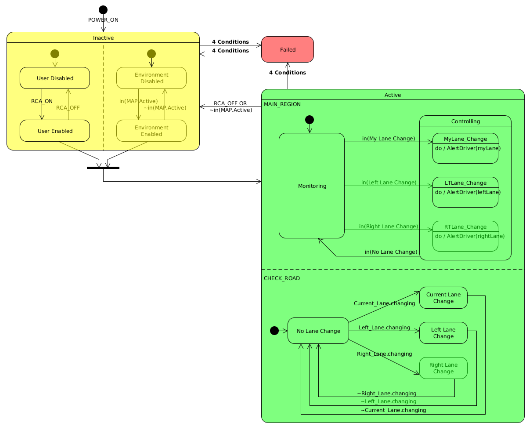


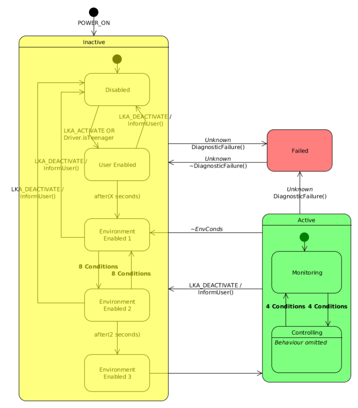


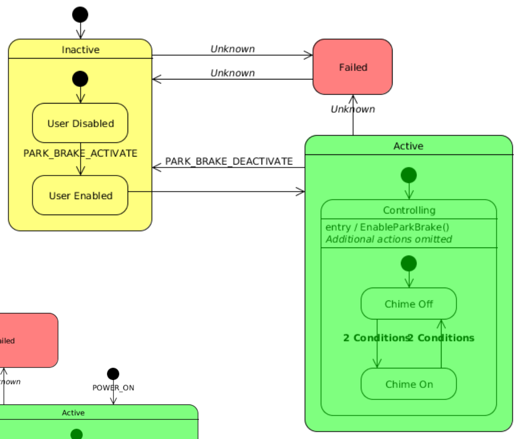
generic feature interface (2)

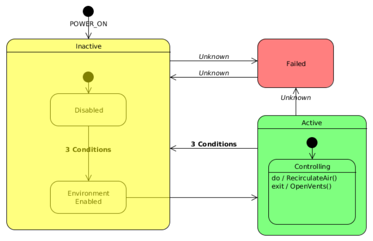


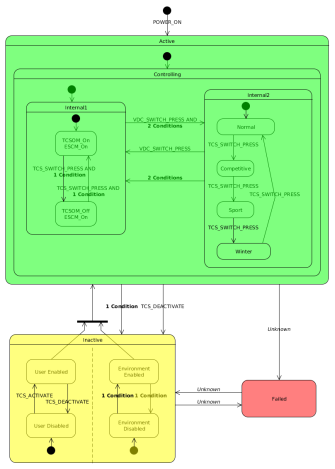


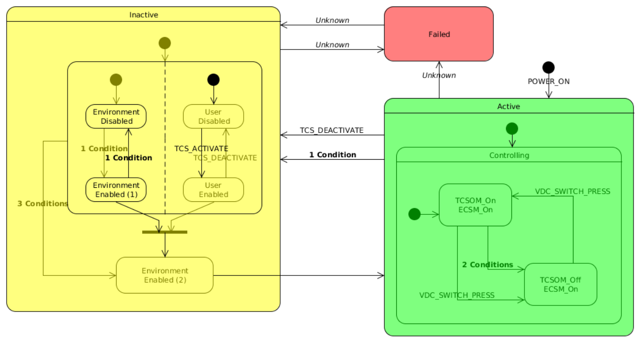












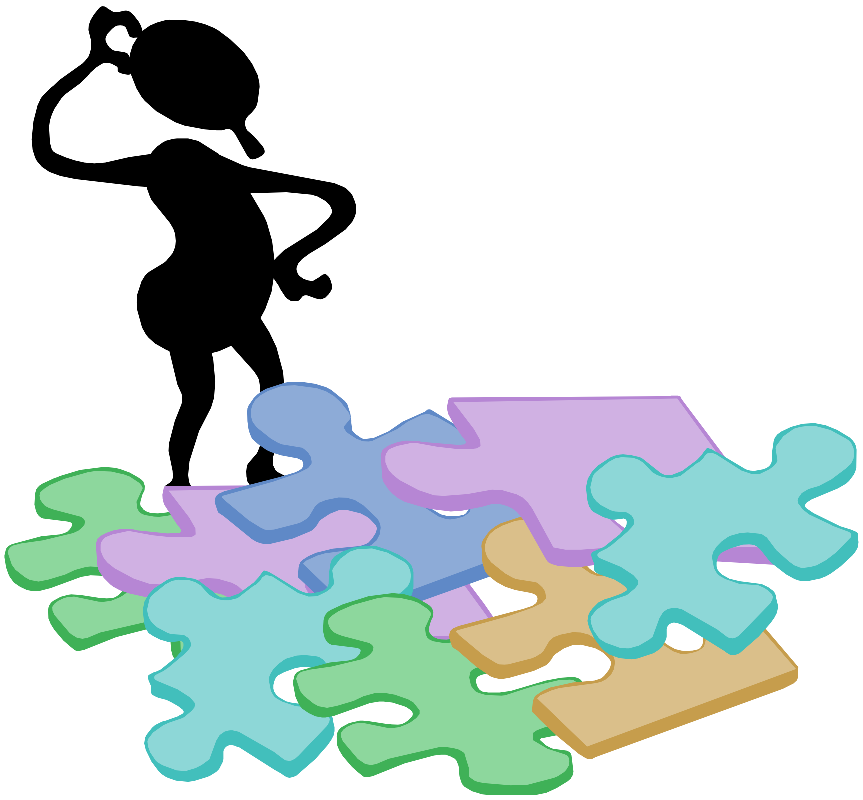
take aways

1. **resolve interactions en masse outside of features**

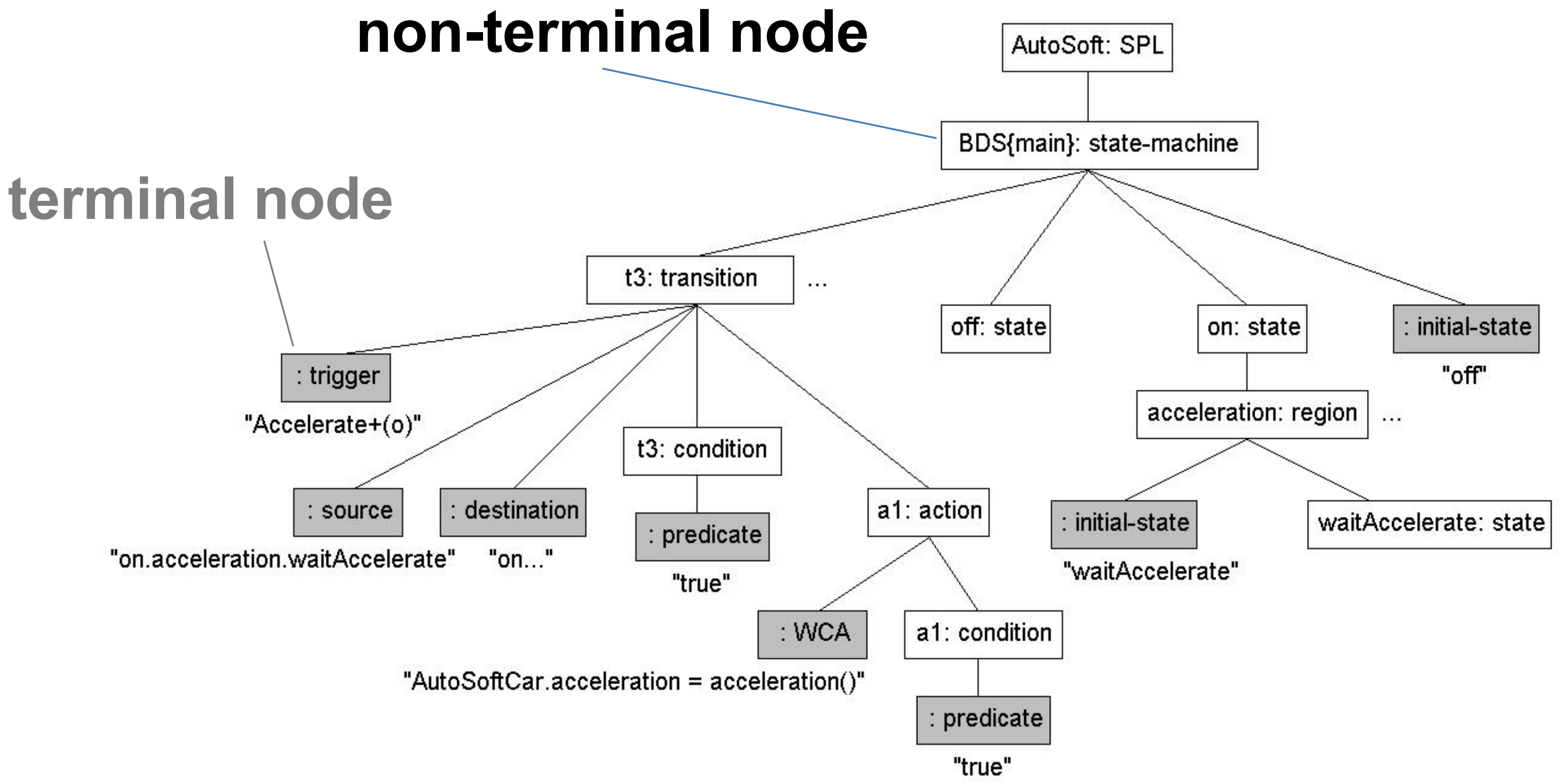
2. **feature modularity to ease complexity, promote parallel development**
   - express fragments wrt extension points
   - explicate intended interactions

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

feature composition



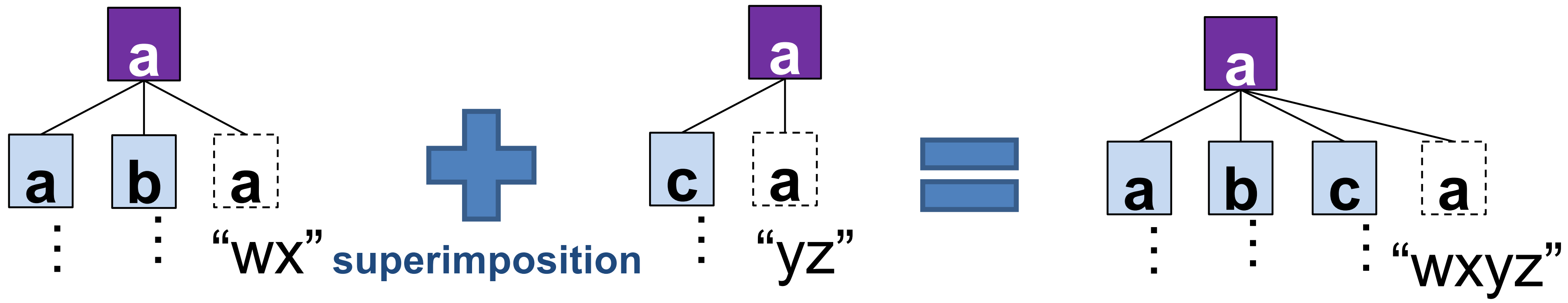
feature structure trees (FSTs)



BDS FST

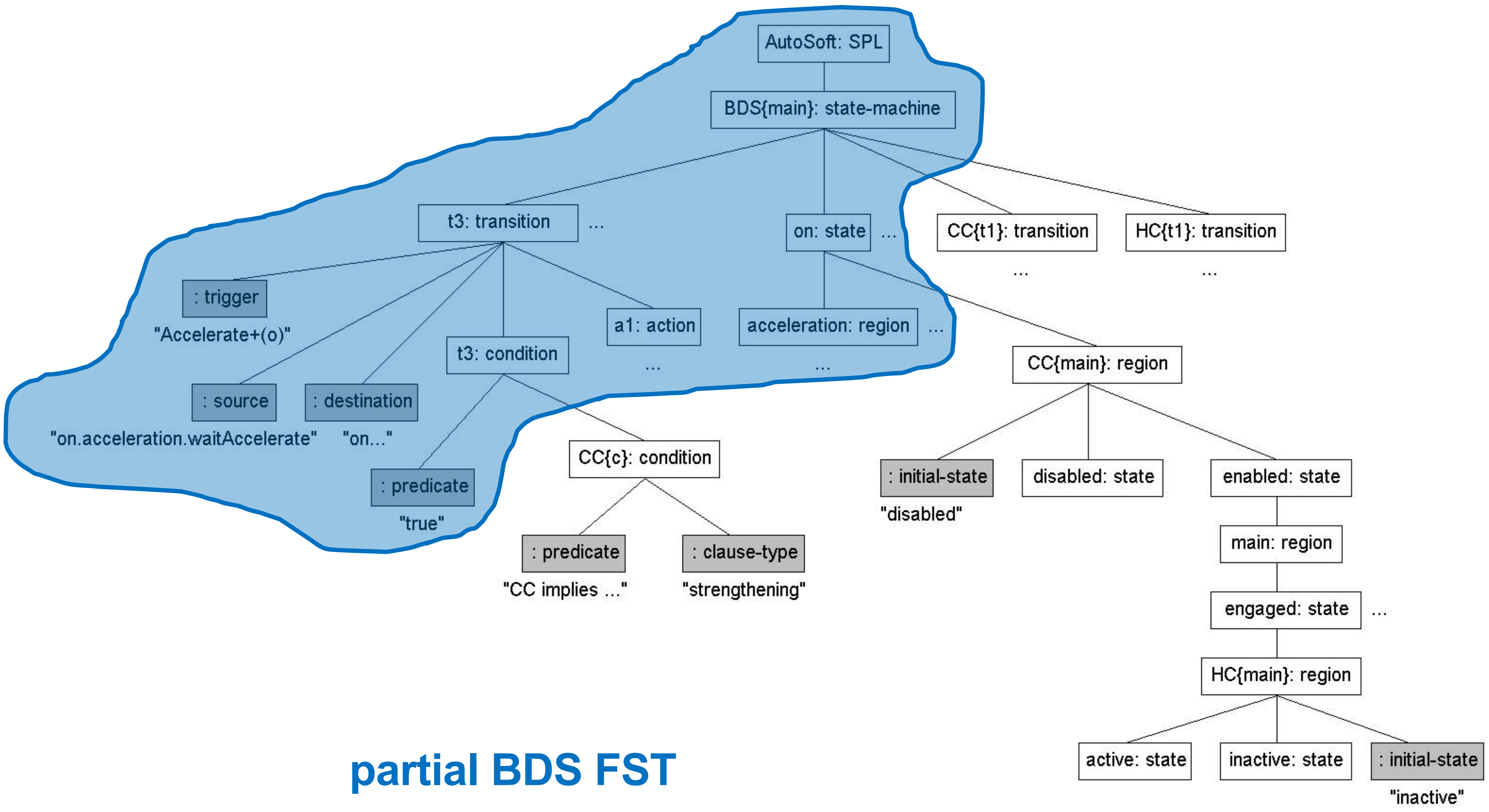
superimposition

compose feature modules by superimposing their feature structure trees (FSTs)



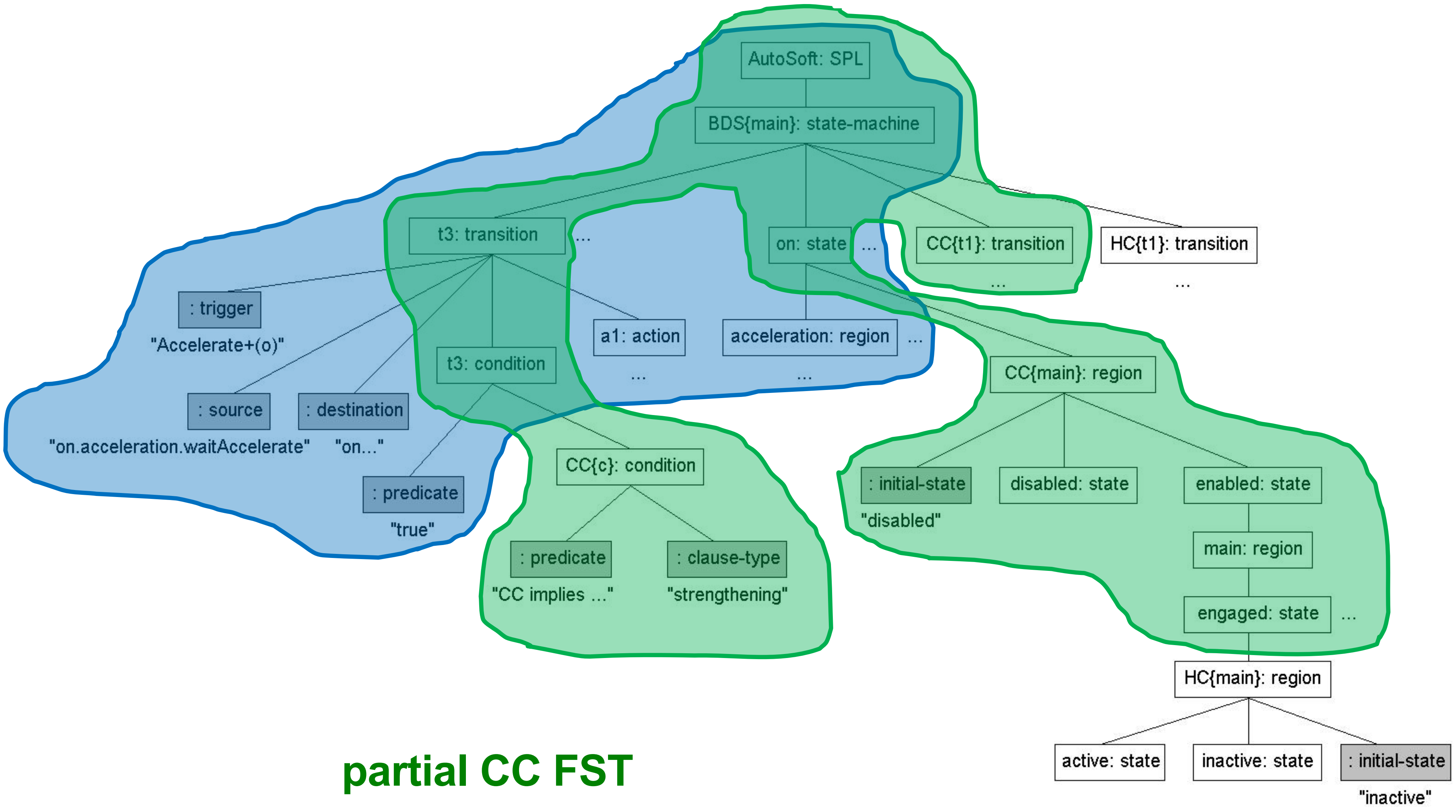
composition is commutative and associative because terminal nodes are not merged

superimposition



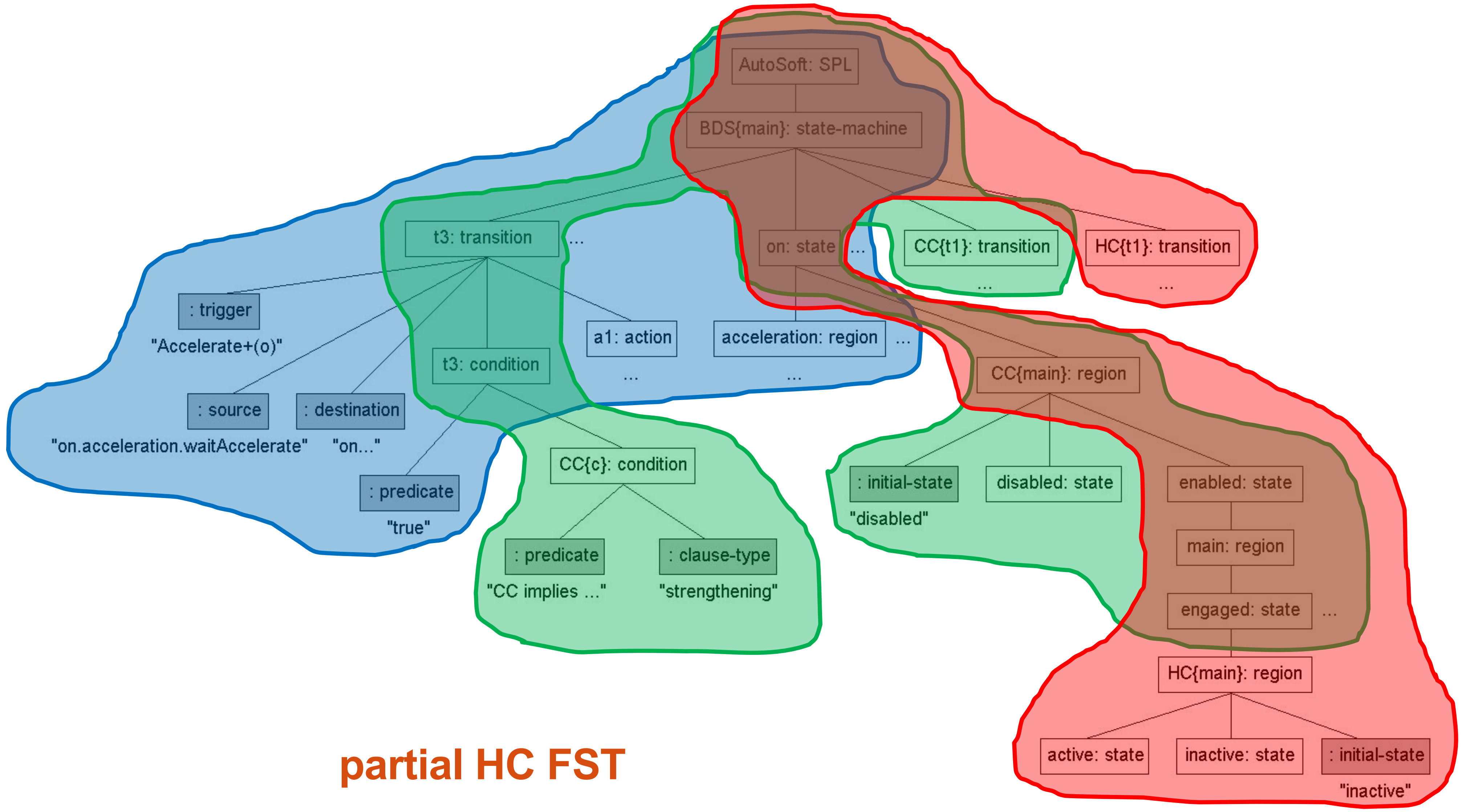
partial BDS FST

superimposition



partial CC FST

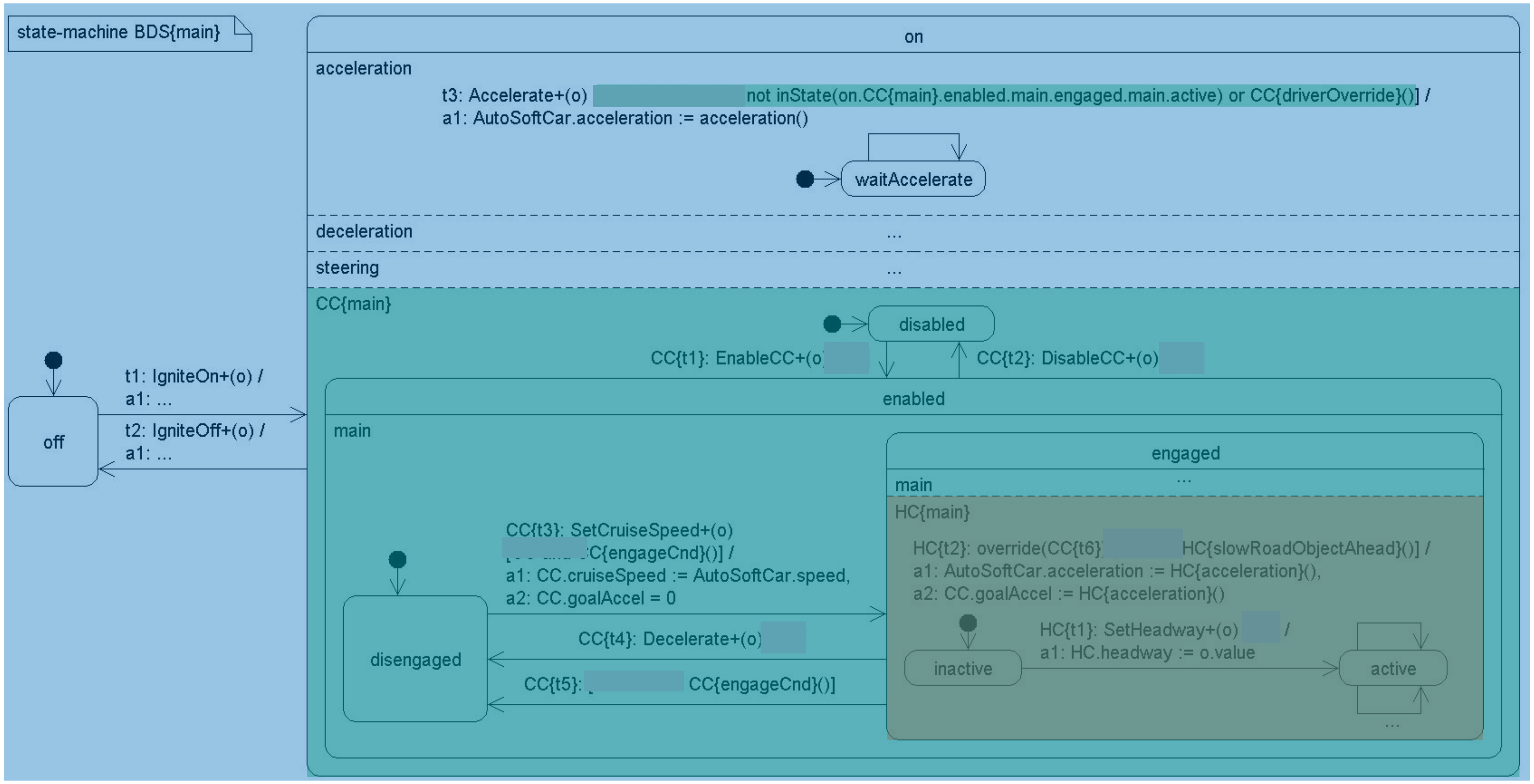
superimposition



partial HC FST

resulting composition (product)

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



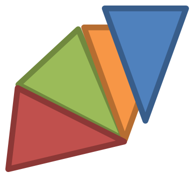
commutativity

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

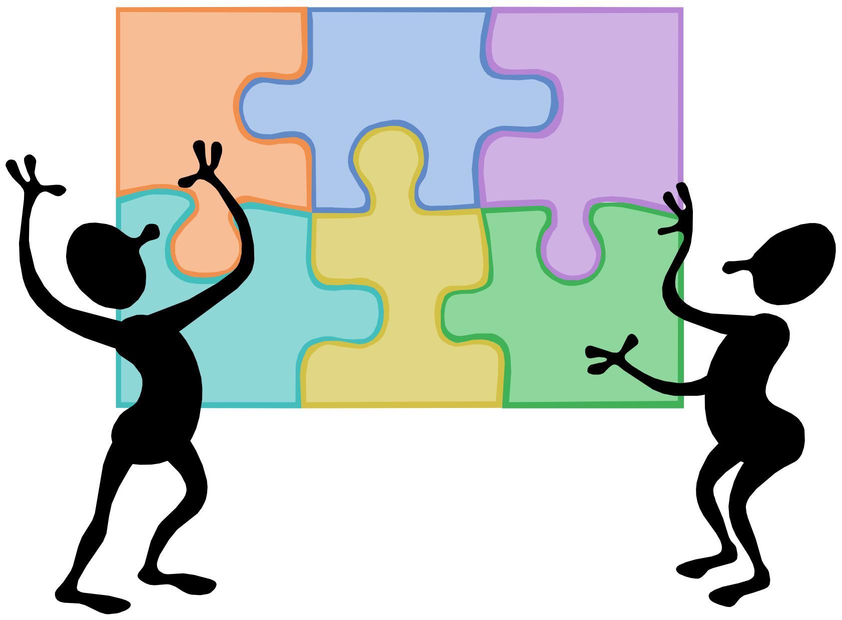
1. resolve interactions en masse outside of features

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

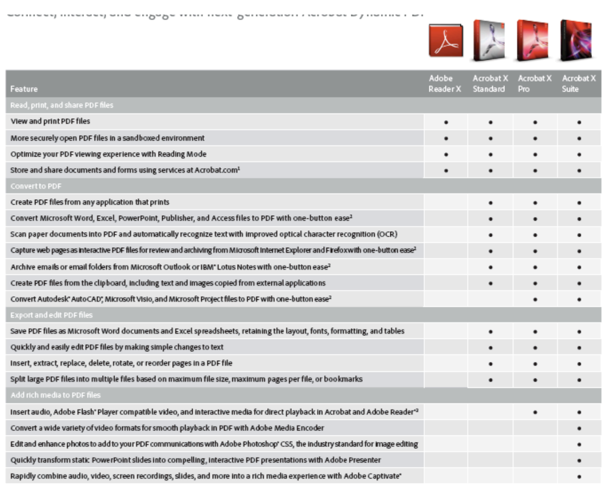
3. feature (public) interfaces hide implementation details
   - expose feature’s inputs/outputs, accessors, mutators
   - generic interface exposes behaviour modes

4. **commutative composition**

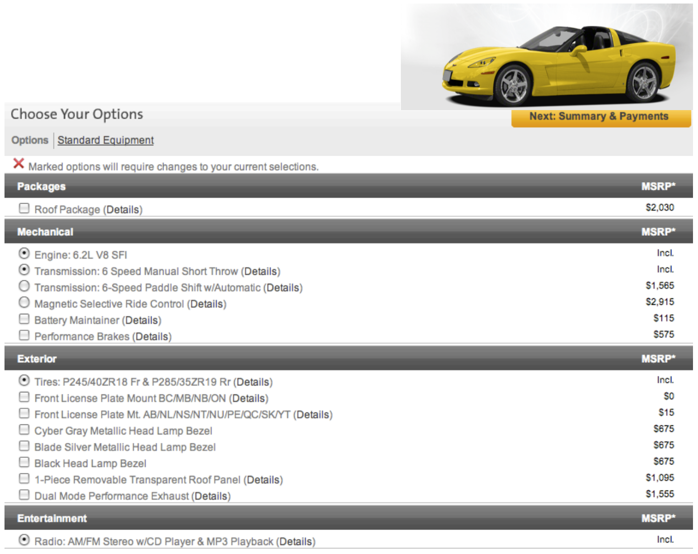
resolving unintended interactions



feature coordination



Feature	Adobe Reader X	Acrobat X Standard	Acrobat X Pro	Acrobat X Suite
Read, print, and share PDF files				
View and print PDF files	•	•	•	•
More securely open PDF files in a sandboxed environment	•	•	•	•
Optimize your PDF viewing experience with Reading Mode	•	•	•	•
Store and share documents and forms using services at Acrobat.com[1]	•	•	•	•
Convert to PDF				
Create PDF files from any application that prints		•	•	•
Convert Microsoft Word, Excel, PowerPoint, Publisher, and Access files to PDF with one-button ease[2]		•	•	•
Scan paper documents into PDF and automatically recognize text with improved optical character recognition (OCR)		•	•	•
Capture web pages as interactive PDF files for review and archiving from Microsoft Internet Explorer and Firefox with one-button ease[2]		•	•	•
Archive emails or email folders from Microsoft Outlook or IBM® Lotus Notes with one-button ease[2]		•	•	•
Create PDF files from the clipboard, including text and images copied from external applications		•	•	•
Convert Autodesk® AutoCAD®, Microsoft Visio, and Microsoft Project files to PDF with one-button ease[2]			•	•
Export and edit PDF files				
Save PDF files as Microsoft Word documents and Excel spreadsheets, retaining the layout, fonts, formatting, and tables		•	•	•
Quickly and easily edit PDF files by making simple changes to text		•	•	•
Insert, extract, replace, delete, rotate, or reorder pages in a PDF file		•	•	•
Split large PDF files into multiple files based on maximum file size, maximum pages per file, or bookmarks		•	•	•
Add rich media to PDF files				
Insert audio, Adobe Flash® Player compatible video, and interactive media for direct playback in Acrobat and Adobe Reader®[2]			•	•
Convert a wide variety of video formats for smooth playback in PDF with Adobe Media Encoder				•
Edit and enhance photos to add to your PDF communications with Adobe Photoshop® CS5, the industry standard for image editing				•
Quickly transform static PowerPoint slides into compelling, interactive PDF presentations with Adobe Presenter				•
Rapidly combine audio, video, screen recordings, slides, and more into a rich media experience with Adobe Captivate®				•



Choose Your Options

Options | Standard Equipment

✗ Marked options will require changes to your current selections.

Packages	MSRP*
☐ Roof Package (Details)	$2,030

Mechanical	MSRP*
● Engine: 6.2L V8 SFI	Incl.
● Transmission: 6 Speed Manual Short Throw (Details)	Incl.
○ Transmission: 6-Speed Paddle Shift w/Automatic (Details)	$1,565
○ Magnetic Selective Ride Control (Details)	$2,915
☐ Battery Maintainer (Details)	$115
☐ Performance Brakes (Details)	$575

Exterior	MSRP*
● Tires: P245/40ZR18 Fr & P285/35ZR19 Rr (Details)	Incl.
☐ Front License Plate Mount BC/MB/NB/ON (Details)	$0
☐ Front License Plate Mt. AB/NL/NS/NT/NU/PE/QC/SK/YT (Details)	$15
☐ Cyber Gray Metallic Head Lamp Bezel	$675
☐ Blade Silver Metallic Head Lamp Bezel	$675
☐ Black Head Lamp Bezel	$675
☐ 1-Piece Removable Transparent Roof Panel (Details)	$1,095
☐ Dual Mode Performance Exhaust (Details)	$1,555

Entertainment	MSRP*
● Radio: AM/FM Stereo w/CD Player & MP3 Playback (Details)	Incl.

› 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

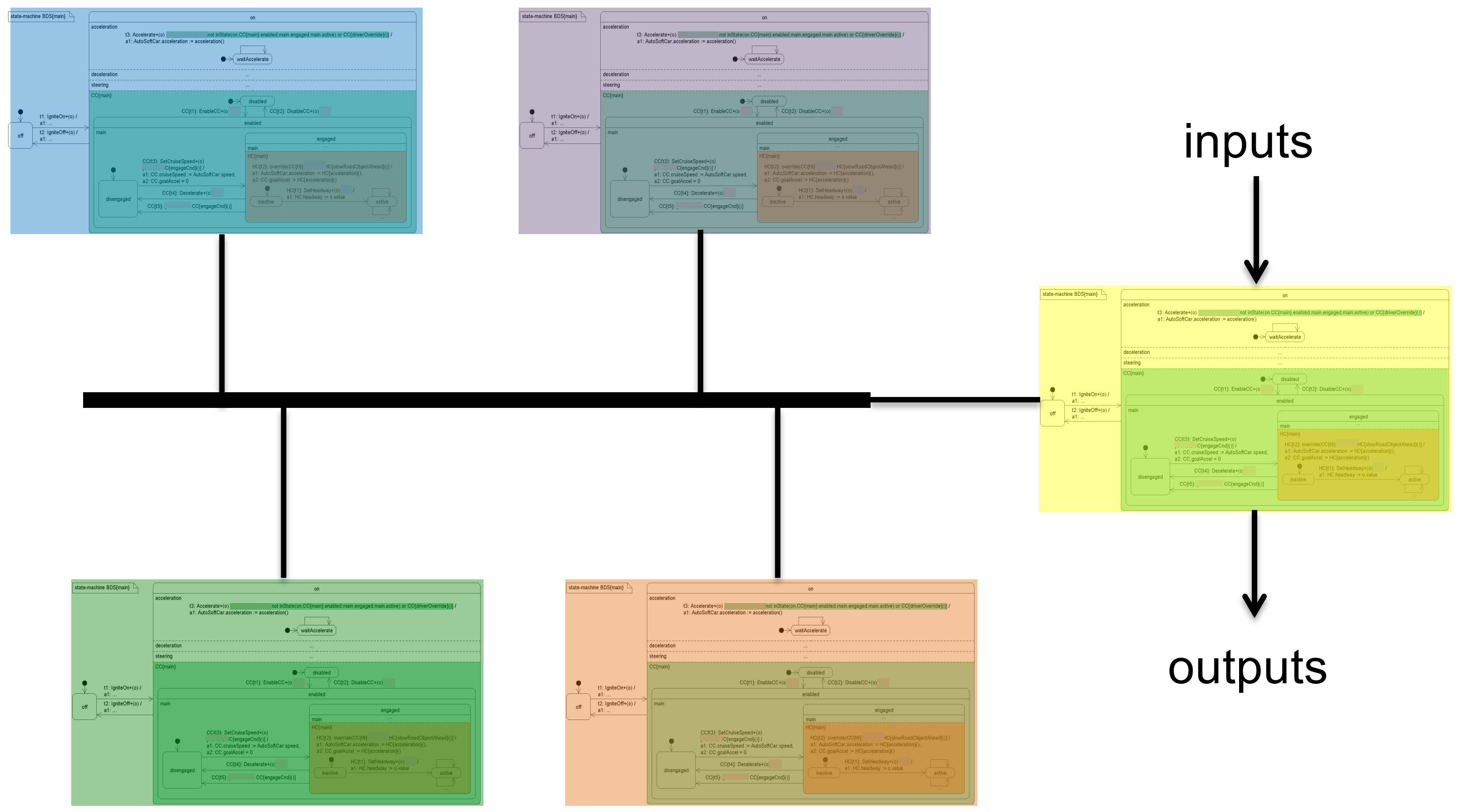
› dynamic integration

› loose coordination

feature coordination

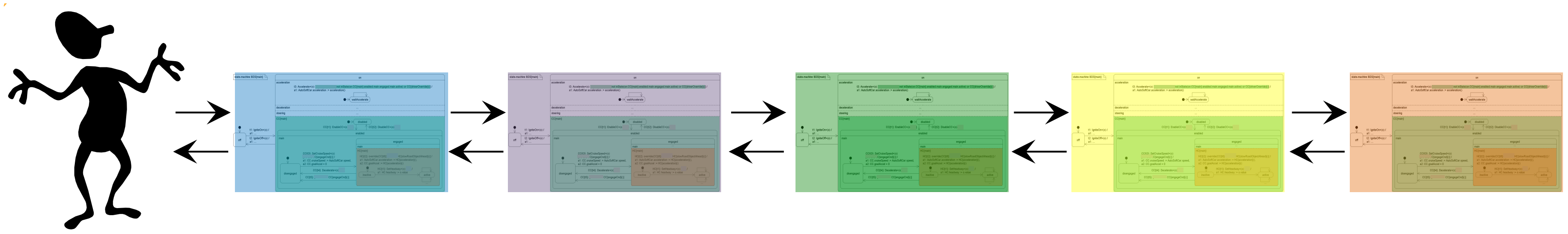
composition is a collection of parallel machines that have been extended with fragments

each machine's interface is simply its inputs and outputs



serializing features

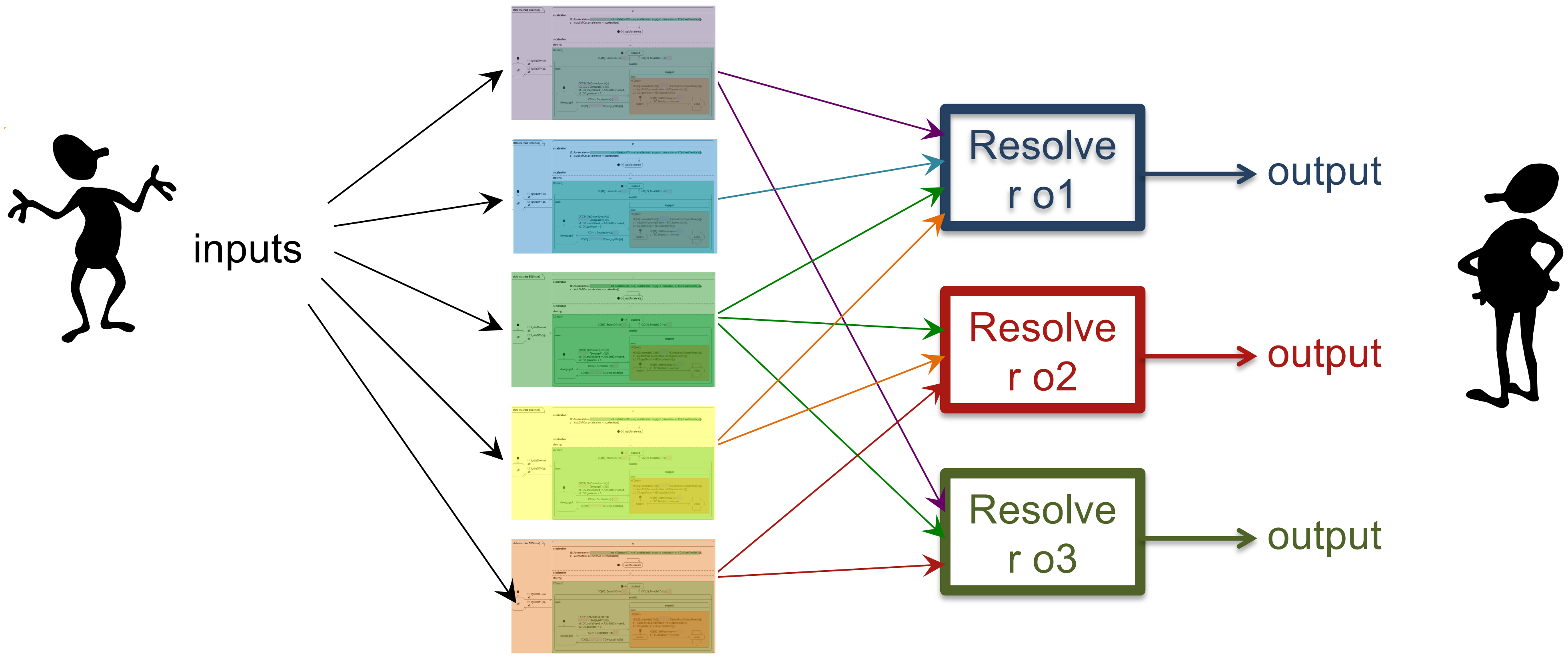
Distributed Feature Composition [Jackson, Zave, TSE'98]



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

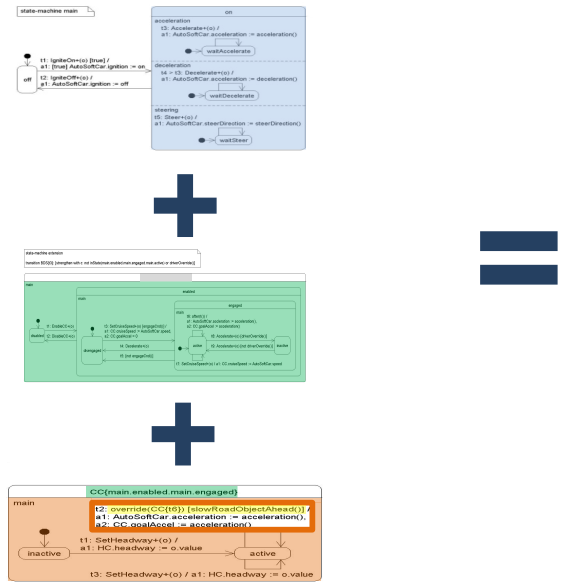
summary

modular features

- extension points
- intended interactions

generic public interfaces

- mode of operation

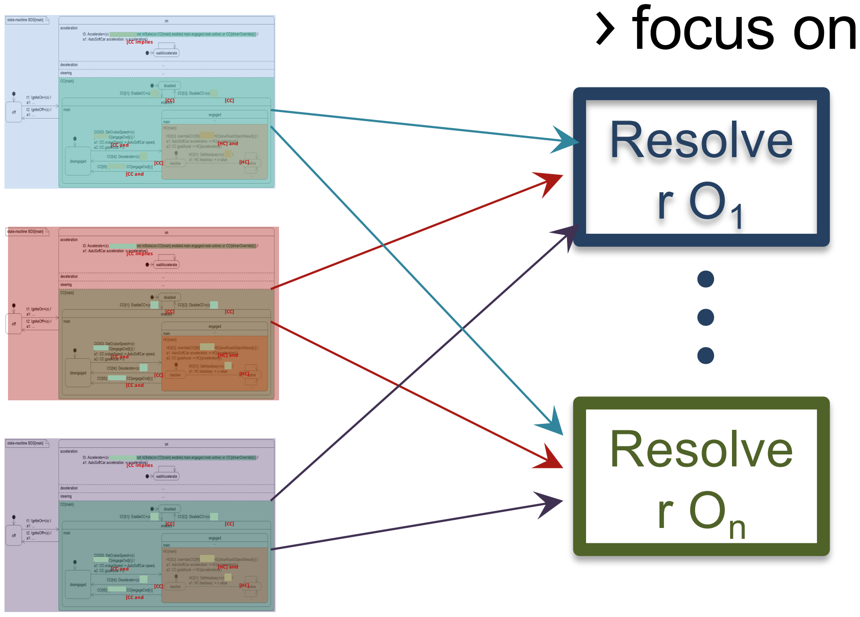

state-machine main
on
acceleration
t3: Accelerate+(o) /
a1: AutoSoftCar.acceleration := acceleration()
waitAccelerate
deceleration
t4 = t3: Decelerate+(o) /
a1: AutoSoftCar.acceleration := deceleration()
waitDecelerate
steering
t5: Steer+(o) /
a1: AutoSoftCar.steerDirection := steerDirection()
waitSteer
t1: IgniteOn+(o) [true] /
a1: [true] AutoSoftCar.ignition := on
t2: IgniteOff+(o) /
a1: AutoSoftCar.ignition := off
off
CC{main.enabled.main.engaged}
main
t2: override(CC{t6}) [slowRoadObjectAhead()] /
a1: AutoSoftCar.acceleration := acceleration(),
a2: CC.goalAccel := acceleration()
t1: SetHeadway+(o) /
a1: HC.headway := o.value
inactive
active
t3: SetHeadway+(o) / a1: HC.headway := o.value

composition

- of feature families
- commutative

coordination

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


Resolve $r\ O_1$
Resolve $r\ O_n$