FEATURE ORIENTED-PROGRAMMING: BACK TO THE FUTURE

Christian Prehofer
Fraunhofer ESK, Munich, Germany,
christian.prehofer@esk.fraunhofer.de

Ludwig-Maximilians-Universität München
Programming and Software Engineering

Source: Universal Studios

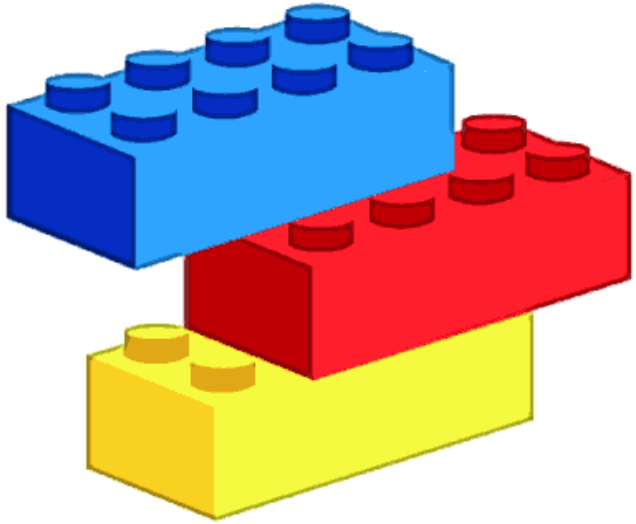
© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Overview

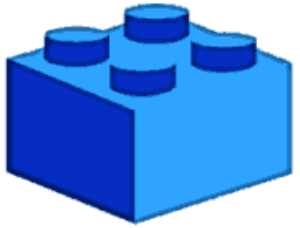
- FOP history and origins
- Feature interactions and feature composition
- Feature-oriented modeling and refinement
- Outlook

© Fraunhofer ESK

How it started …



- "Feature or SW composition" is not a new idea …
  old dream of software
  - Lots of new composition concepts in 90ties
    - Mixins, composition filters, aspects, …
- Feature Interaction WS series - started 92
  - Motivated by specific problems in Telecom
  - Highly-entangled features



- Feature-oriented Domain Analysis – what the user wants
  - And-or tree to structure requirements
- Monads and monad transformers …. more later
  - Powerful theory to express composition & properties as types
  -

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Feature Interaction Example

- Classic Example: Call forwarding and call waiting
  - **Call Forwarding**: forward calls when busy
  - **Call waiting**: interrupt existing call
  - FI Problem: Incoming call while other is active: forward or notify with call waiting
- Notice: Feature interactions are about **system behavior**
- Reminder: in real systems, we have dozens or **hundreds of features**
- Problem: Modularity in specification, design, implementation and composition

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Multi-Feature Interactions

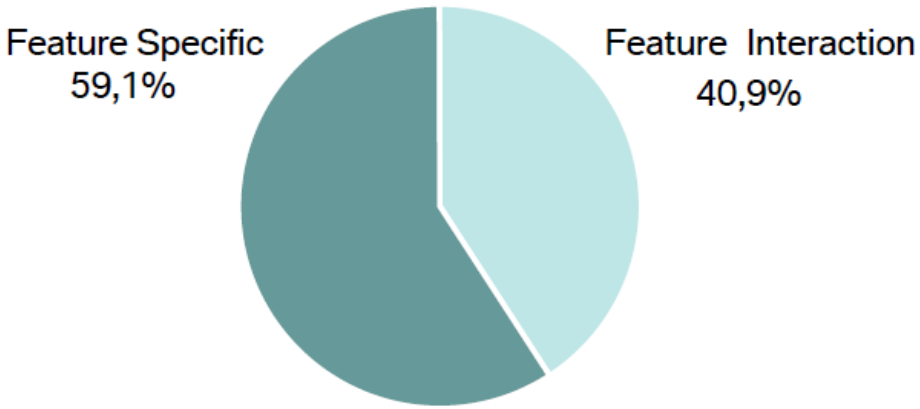
Example:

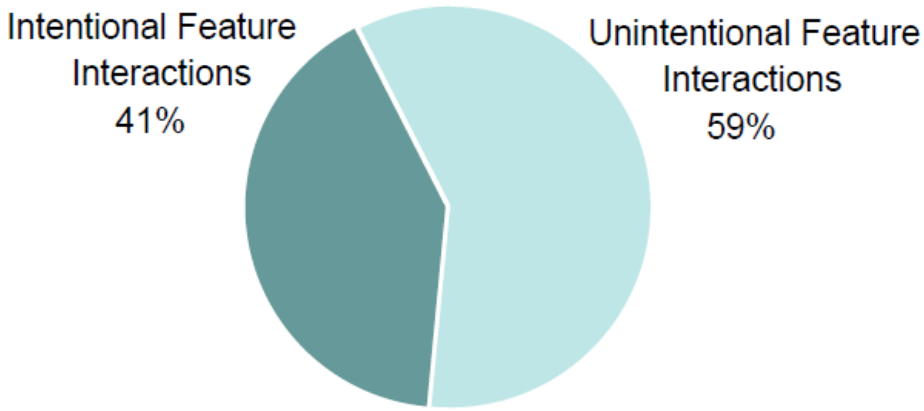
1. Basic call
2. Call waiting: Take incoming call and put first call on hold
3. Lock phone (by lock key) – disable keys

- Interaction between (1) and (3) – solved by “exception”
  - Calls can be taken when phone is locked – one button is unlocked
- 3-feature interaction happens now
  - Call waiting active while phone is locked
  - Second call is announced, but cannot be taken as only one button unlocked

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Feature Interactions – not just a Telecom Problem





Feature Interactions in Automotive Infotainment.
Source: S. Benz, Generating Tests for Feature Interaction
http://mediatum2.ub.tum.de/node?id=805656

"Feature interaction is a major cause of system failures, and its avoidance is a major cost for system administrators deploying new features"
Dobson, Simon, Sterritt, Roy, Nixon, Paddy and Hinchey, Mike (2010)
Fulfilling the Vision of Autonomic Computing. IEEE Computer, 43 (1). pp. 35-41.

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

FOP vs AOP

Aspect-oriented Programming

- Focus on Syntax, Modularity defined as **code modularity**
  "Typically, an aspect is *scattered* or *tangled* as code, making it harder to understand and maintain." (Source: Wikipedia)

- AOP appeared round 1997, Semantics published 2001-2004

- Aspect interference/interactions difficult to define
  - Interaction is about *semantics*
  - Use only one aspect at a time

© Fraunhofer ESK

Overview

- FOP history and origins
- Feature interactions and feature composition
- Feature-oriented modeling and refinement
- Outlook

© Fraunhofer ESK

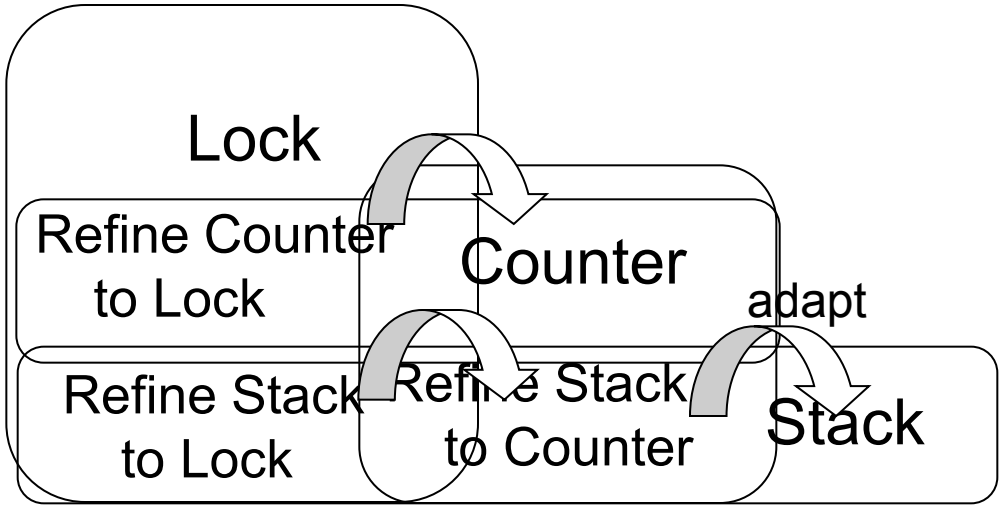
Feature-oriented Programming

Components are built by composing features

- Features have a base class, e.g. Stack, Counter, Lock

Adaptors are used to glue components together

- Adapt functionality
- Resolve feature interactions

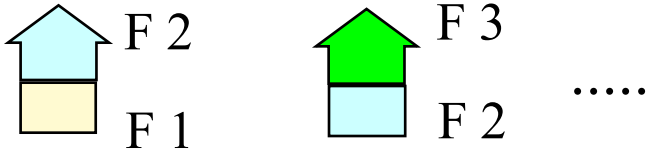


© Fraunhofer ESK
www.esk.fraunhofer.de/EN

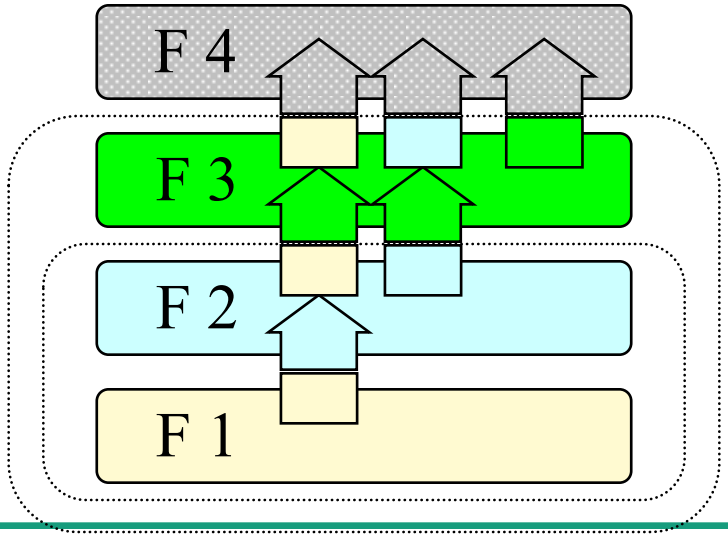
Feature Composition Architecture

Features (State + Methods)

- Adaptors/lifters
  - Adapt one feature to the context of another one



- Composition architecture
  - Origin: Monad compositon with „lifters“



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

My first “monadic” FOP Program - 1996 in Gofer

```
instance StackMonad (StackT [Int] m)
     where
   push a    = do{ s <- get;
                   put (a:s) }
   pop       = do{ s <- get;
                   put (tail s);
                   result (head s)}
   is_empty = do{ s <- get;
                   result (s==[]) }
```

```
instance CountMonad (CountT Int m)
        where
    size   = get
    inc    = do{ i <- get;
                 put (i+1) }
    dec    = do{ i <- get;
                 put (i-1) }
```

```
instance (StackMonad m,
        CountMonad (CountT Int m)) =>
        StackMonad (CountT Int m)
         where
   push a    =  do{ inc ;
                    lift (push a)}
   pop       =  do{ dec ;
                    lift pop}
```

Note: Lifters transform the type

„From Inheritance to Feature Interaction or Composing Monads“, 1997.

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Semantic Feature Composition or Semantic Refinement

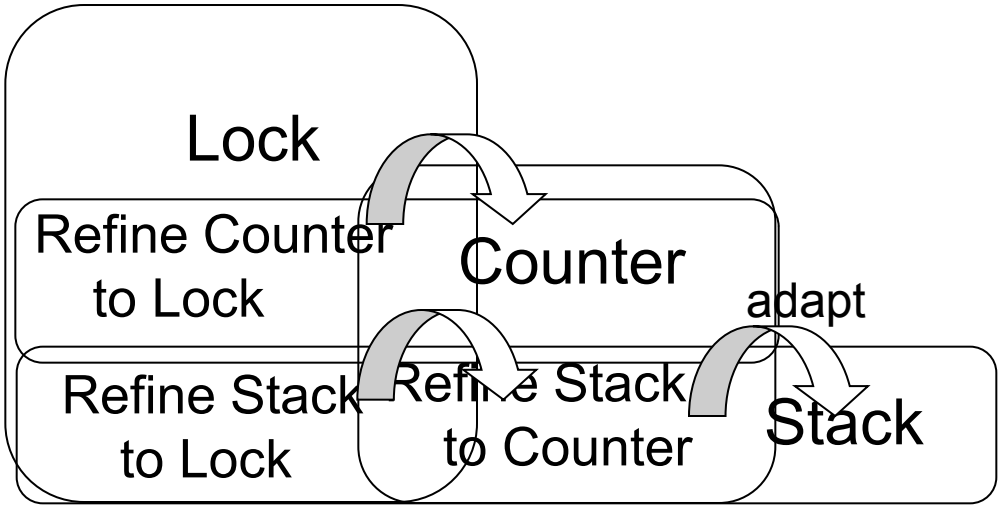
If we add a feature B to a feature A, will feature A still behave in the same way?

- Ok for “harmless” features which only adds extra functionality
- Original behavior on state of feature A (instance) is maintained

Examples

- Stack + Counter $=_{Stack}$ Stack
- Stack + Counter + Lock $=_{Stack}$ Stack

  if Lock is unlocked



$exp1 =_F$ exp2 **semantic equality** of two expressions on state of feature F
I.e. state of F is identical after executing exp1 and exp2

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Harmless Features

Goal is a calculus to reason about combination of features and composition

A feature D is called **conservative** wrt a feature F if

$$D * F =_F F$$

Also called **„harmless“** feature.

```
void push(char a) {
    inc() ;
    s=String.valueOf(a).concat(s);
}
```

$=_{Stack}$

```
void push(char a) {
    s=String.valueOf(a).concat(s);
}
```

Notion of semantic refinement first noted as “semantic inheritance” or “behavioral subtyping”, G. Leavens, 1996

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

The problem of (Multiple) Feature-Compositions

Multi-feature Composition: A + B + X

Modularity Problem:
If we know the effect of feature A on X and of feature B on X,
what can we conclude about adding both A and B to X?

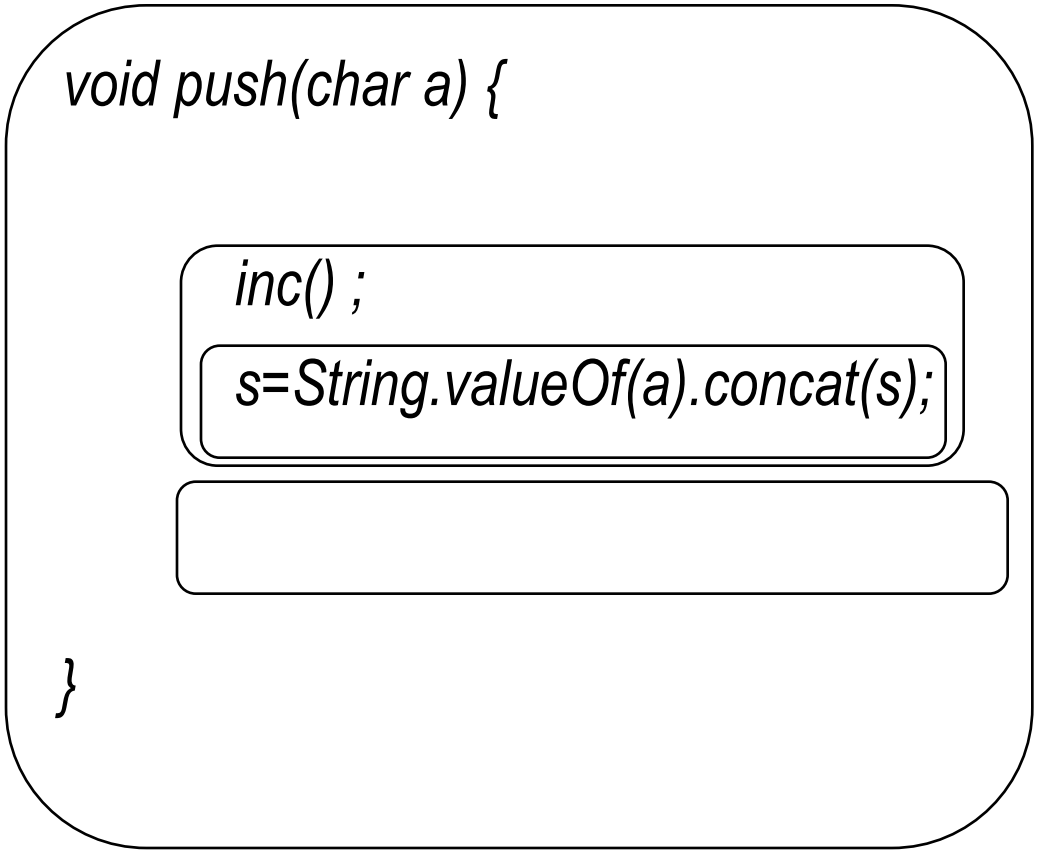
- E.g. what do we know about Stack + Counter + Lock
- Typical problem if you compose multiple features (from different sources)

Question: Are “harmless features” compositional?

Same for AOP and “harmless advice”

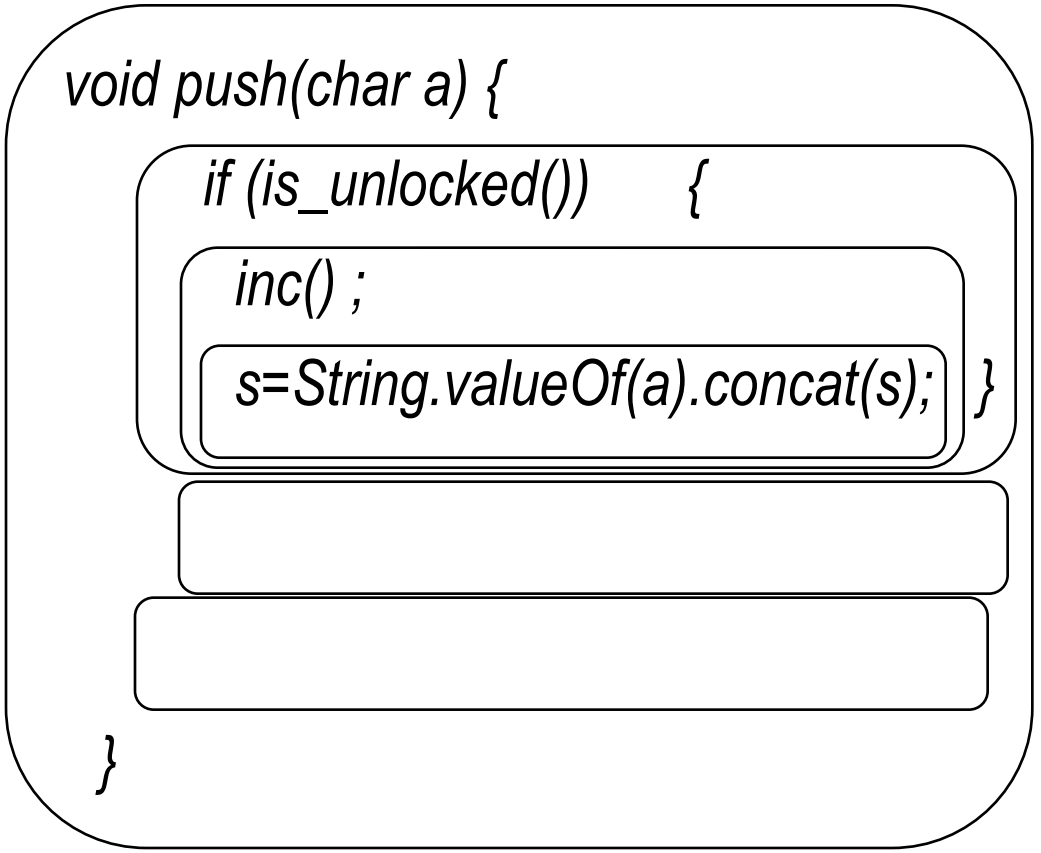
© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Feature Composition Problem: Stack + Counter



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Feature Composition Problem: Stack + Counter + Lock



Lock

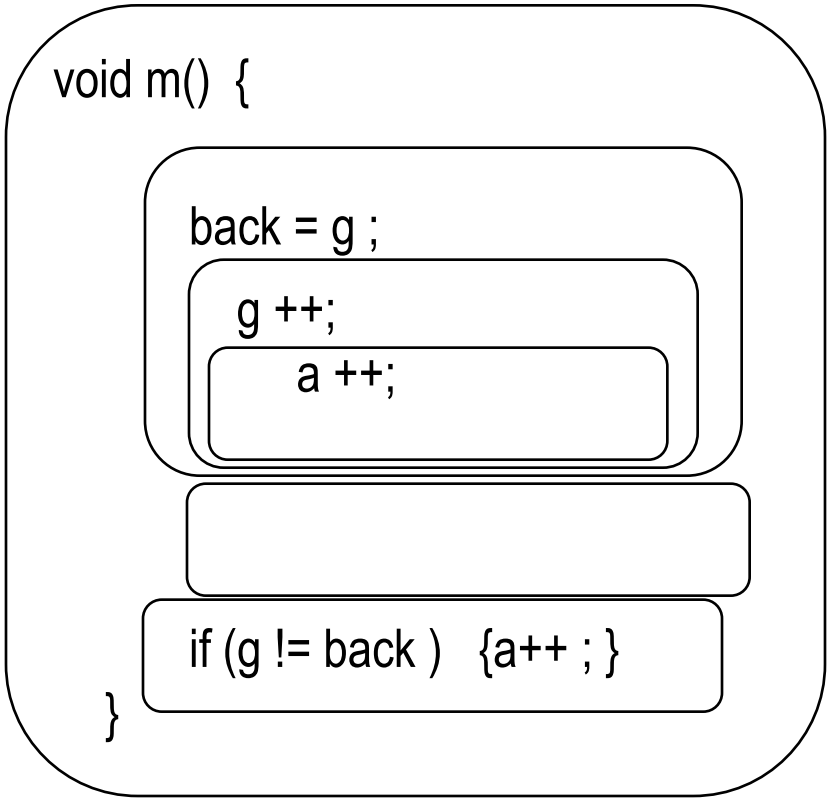
Counter

Base Method

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Fraunhofer ESK

Composing „harmless“ Features is not harmless



Feature F

Feature G

Feature X

Feature F

```
class X {    int a = 0;
      void m()  { a ++; }  }
class G  { int g = 0;  }
class F  { int back;  ….  }
refines X to G {
  void m() { g ++;
      super.m() ; } }
refines X to F uses G {
  void m()  {
                back = g ;
                 super.m() ;
     if (g != back )
        {a++ ; } } }
```

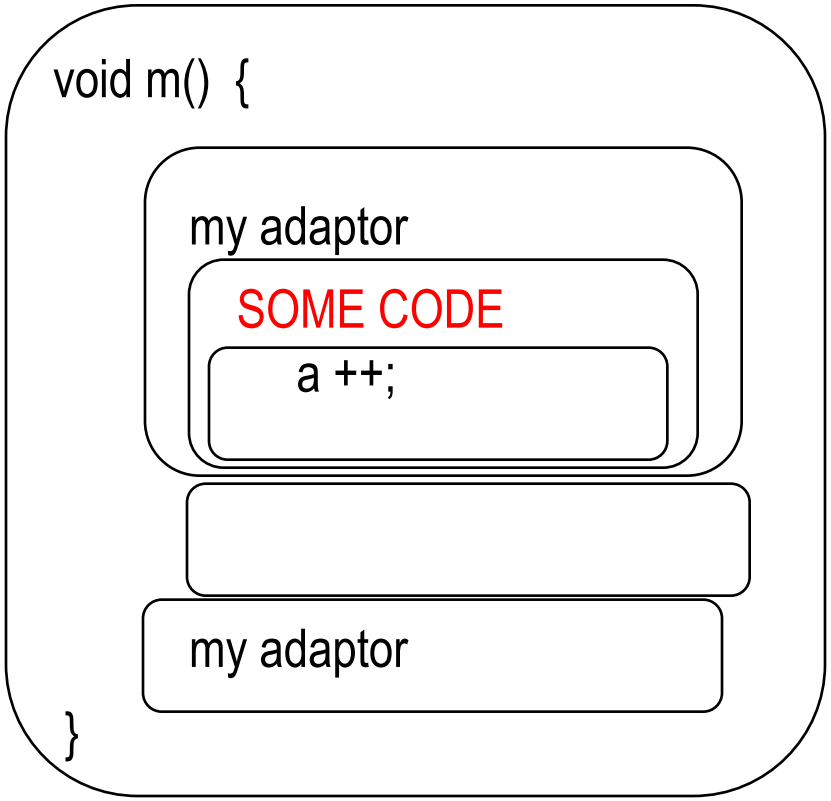
*Features F and G are harmeless wrt X, but F * G is not!*

F has „hidden“ dependency on G

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Fraunhofer ESK

How to write „really harmless features“



D is **independent** of *D'* over a feature *X*, if *D* is conservative over the random V-extension of *x()* for all methods *x()* of *X*, where *V* is the set of variables which are modified by *D'*.

Theorem: A composition *D* + *D'* of is a conservative extension of a feature *X* if the following holds:

- *D* and *D'* are conservative over *X*
- *D* is independent of *D'* over *X*

To write a really harmless adaptors/advice, assume that the code is already modified by other adaptors!

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

In search of a good formalization for features & composition

- Monads as a foundation for Features & Composition
  - Monads are state-transformers – type describes stateful effect
  - Motivated by Liang 95
- Feature Composition as Monad compositions
  - Establish semantic properties on composition by types
  - Prehofer 99, Oliviera 2010

Theorem 1 (Harmless Advice) *Consider any base program bse and any advice adv with the types:*
$bse :: \forall t.(MonadTrans\ t, Monad\ (t\ \kappa)) \Rightarrow Open\ (\alpha \rightarrow t\ \kappa\ \beta)$
$adv :: \forall m.(Monad\ m, Monad\ (\tau\ m)) \Rightarrow Augment\ \alpha\ \beta\ \gamma\ (\tau\ m)$
where κ is a monad and τ a monad transformer. If a function $proj :: \forall m, a.Monad\ m \Rightarrow \tau\ m\ a \rightarrow m\ a$ *exists that satisfies the property:*

$$proj \circ lift \equiv id$$

, then advice adv is harmless with respect to bse:

$$proj \circ (weave\ (adv \circledast bse)) \equiv runIdT \circ (weave\ bse)$$

References

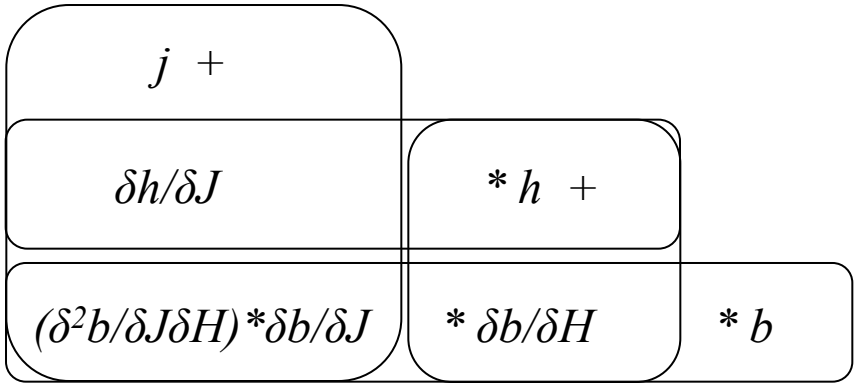
- Prehofer 1997, 1999
- Oliveira, B. C., Schrijvers, T., and Cook, W. R. EffectiveAdvice: disciplined advice with explicit effects. AOSD '10.
- Liang, S., Hudak, P., and Jones, M. Monad transformers and modular interpreters. POPL 95

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Formalization of features & composition: Differential Calculus

„Differential Calculus“ to describe (syntactic) combination of features

- Adaptor of *h* to a feature *G* is denoted as *differential δh / δ G*
  - *Nice to express multiple feature interactions*
  - *But does not provide semantics to features*



Reference

J. Liu, D. Batory, and S. Nedunuri. 2005
Modeling Interactions in Feature Oriented Designs,

© Fraunhofer ESK

Overview

- FOP history and origins
- Feature interactions and Feature Composition
- Feature-oriented modeling and refinement
- Outlook

© Fraunhofer ESK

Feature-oriented modeling and refinement

- Goal is modularity for statechart diagrams
  - Cross-cutting state diagrams into features and interactions
  - Semantic refinement concepts for adding of features

Main benefits of graphical combination of features

- Reduce size & complexity of grahical specifications
- Compose only models of desired features
- Features as consistent design concepts in requirements, design and implementation

© Fraunhofer ESK

Example: Email Features

Features of an Email System

- Encryption of Emails
- Forwarding Emails
- Auto-reply
- Filtering of Emails
- Virus scanner
- ...

Feature Interactions

- Encryption and Forwarding
  - forward only encrypted
- Encryption and Auto-reply
  - Titel of email sent in plain as reply

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Component-Design with Statecharts

Statechart describes behavior of an object

Transitions triggered by external function call or internal action

called_function() [condition] / action

Composition of statecharts from features

Object specification:	statechart
Feature:	partial statechart
Interaction handler (adaptor)	partial statechart
Feature composition:	statechart refinement

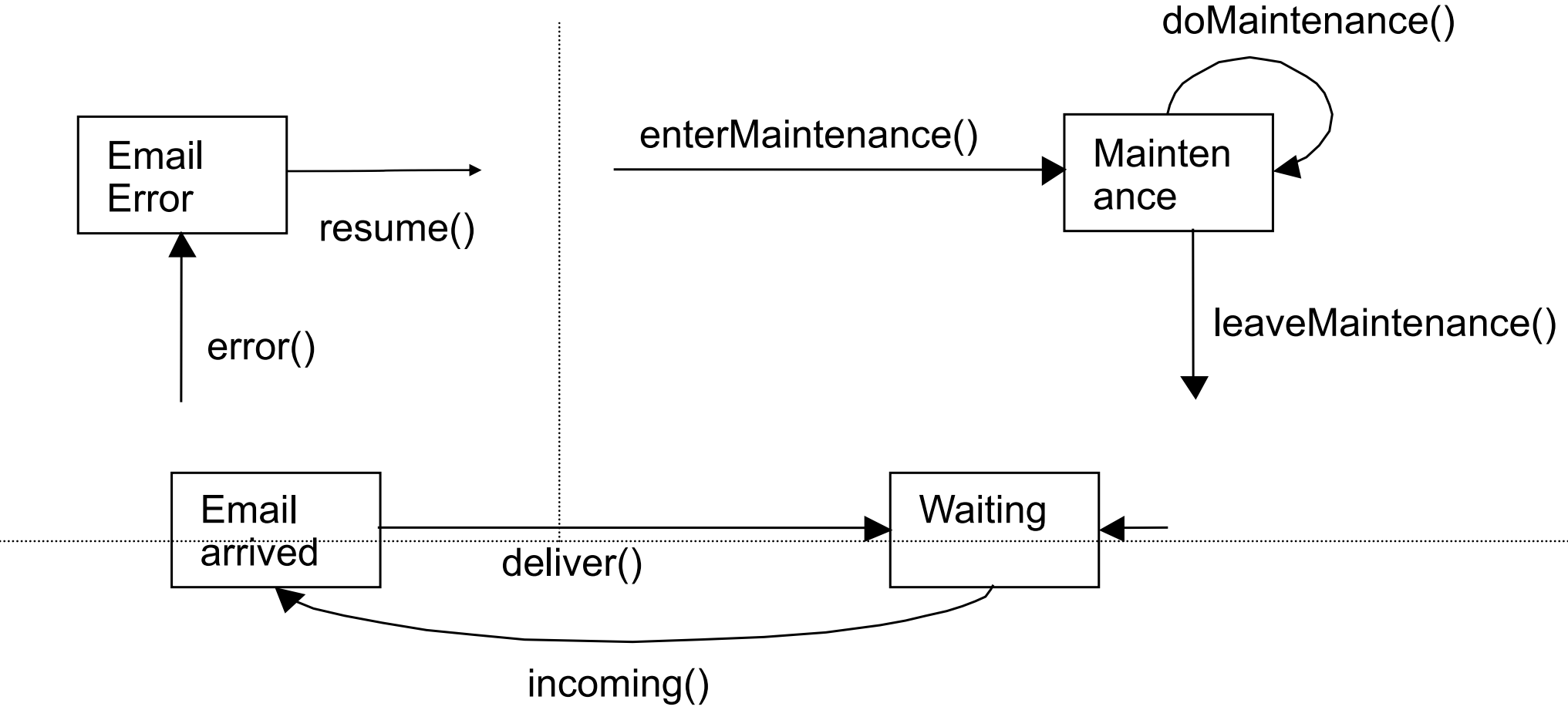
Modular development of statecharts from features

- Feature-interactions as statecharts-refinement

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Feature model with partial statecharts

Example: 3 features which add states



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

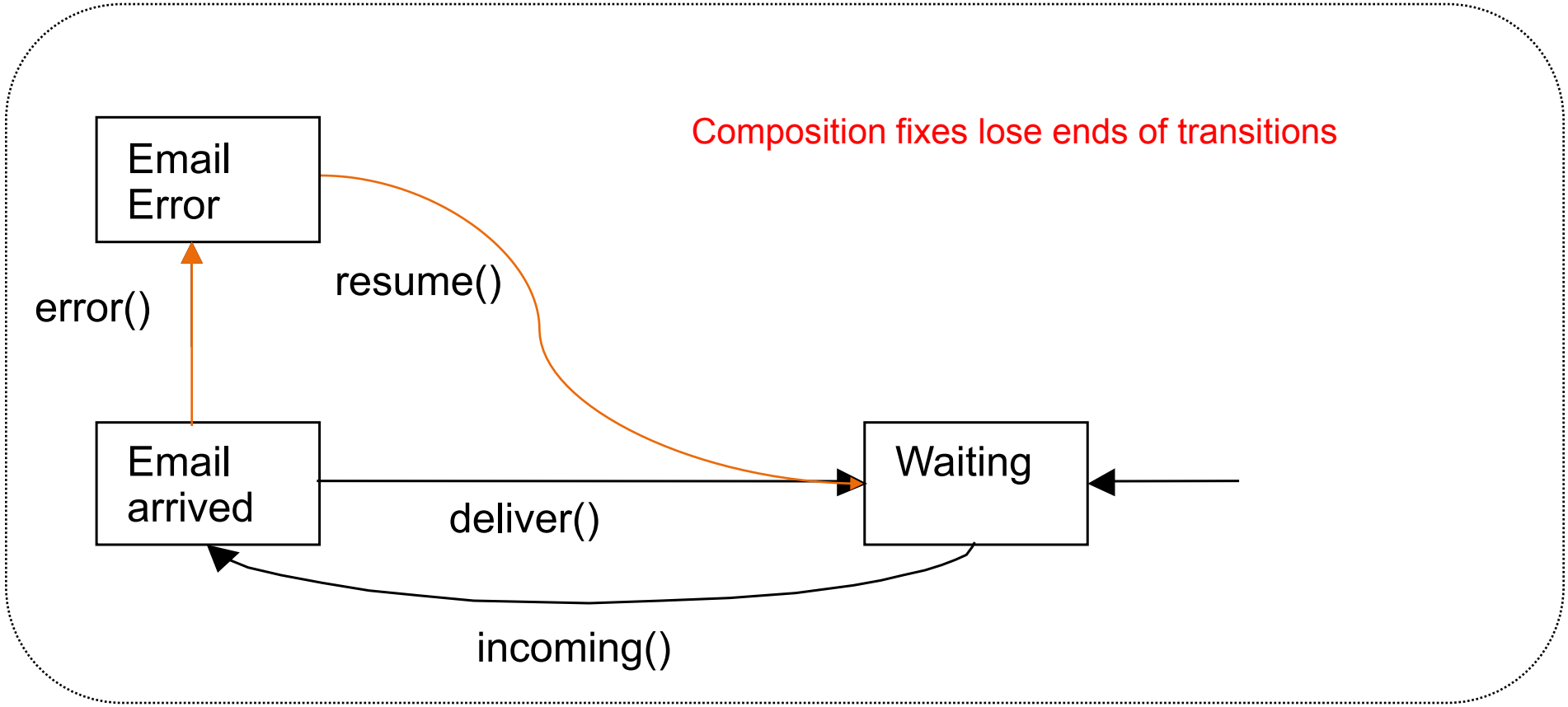
Classes of Features/Statecharts

1. Base features with a complete statechart
   - includes an initial state.
2. State-oriented features with a partial statechart
   - some states and transitions, but initial/final states not required
   - E.g. feature “MaintenanceMode” with one new state
     - Only reachable by new transitions from other features.
3. Transition-oriented features which define transitions
   - No (persistent) state

- Interaction specification („adaptors“) according to these classes
- Combination rules according to these classes

© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Combination of 2 Features



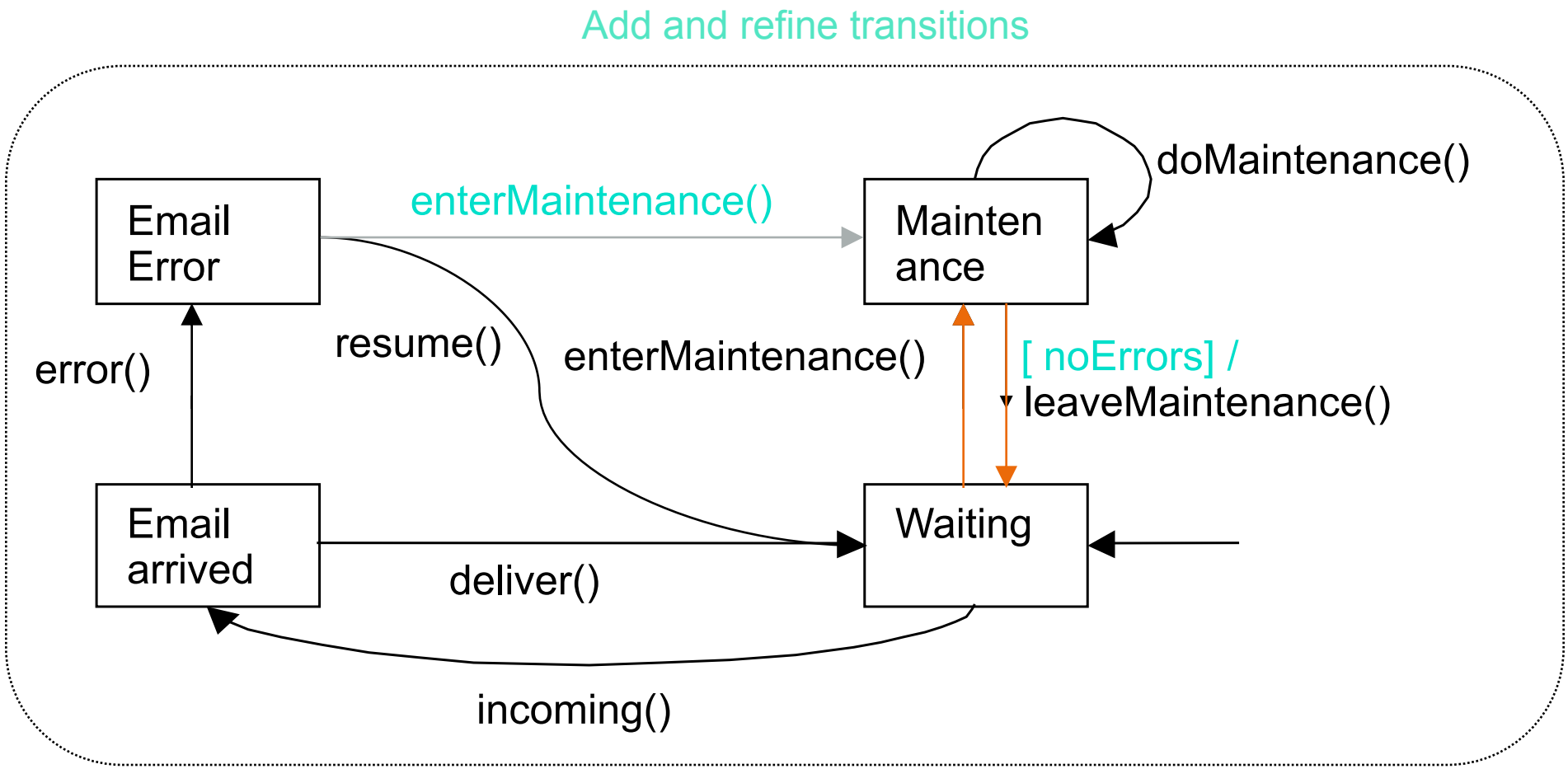
© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Composition as Semantic Refinement

- „Loose“-Semantics based on external view of traces
  - Specifies input/output behavior („black box view“)
  - Semantics of a statechart are all possible or non-conflicting traces
    - If unspecified action occurs, anything is possible („chaos“)
- Refinement adds specific details
  - Reduces the number of possible traces
  - Behavior is compatible with original statechart (subset or original trace set)
- Refinement steps for statechart diagrams
  - Extend the number of states
    - Add new states and „new“ transitions
  - Refinement of transitions
    - Refine transition by statechart with internal transitions only

© Fraunhofer ESK

Combination of 3 Features

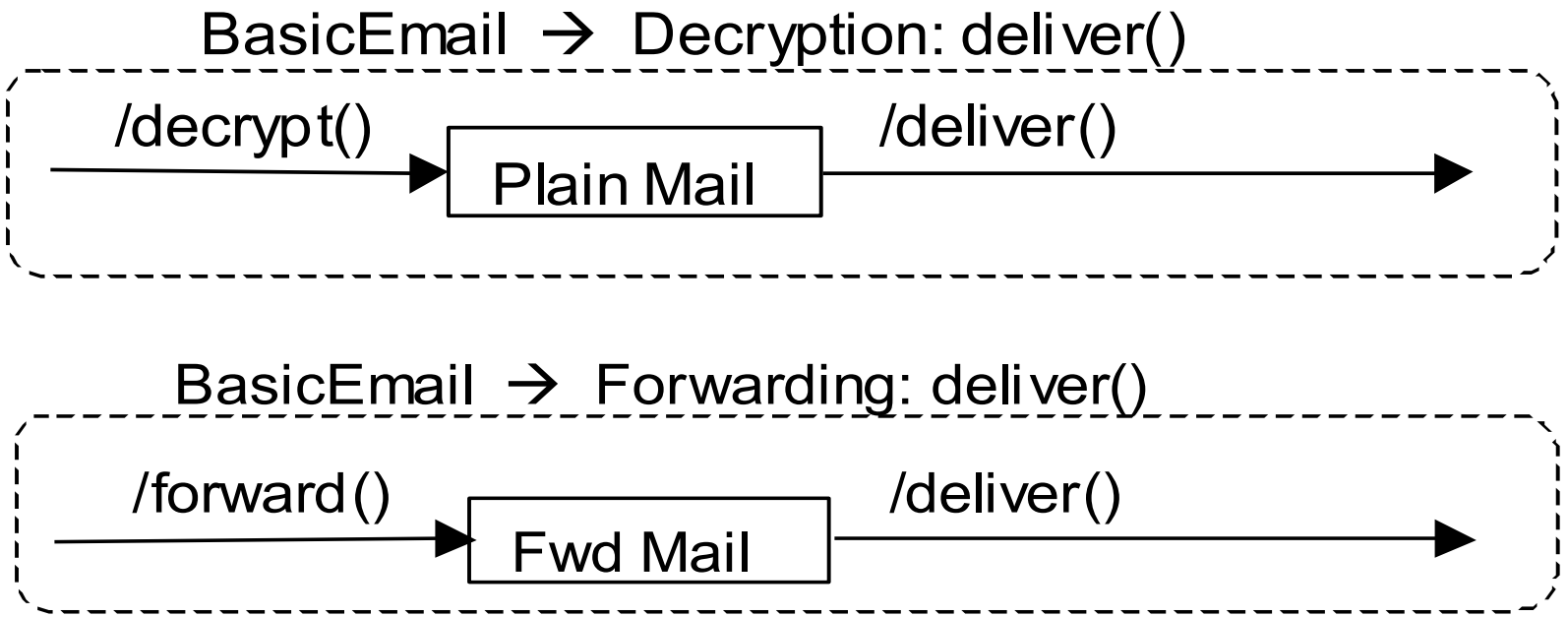


© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Combination Rules for Features

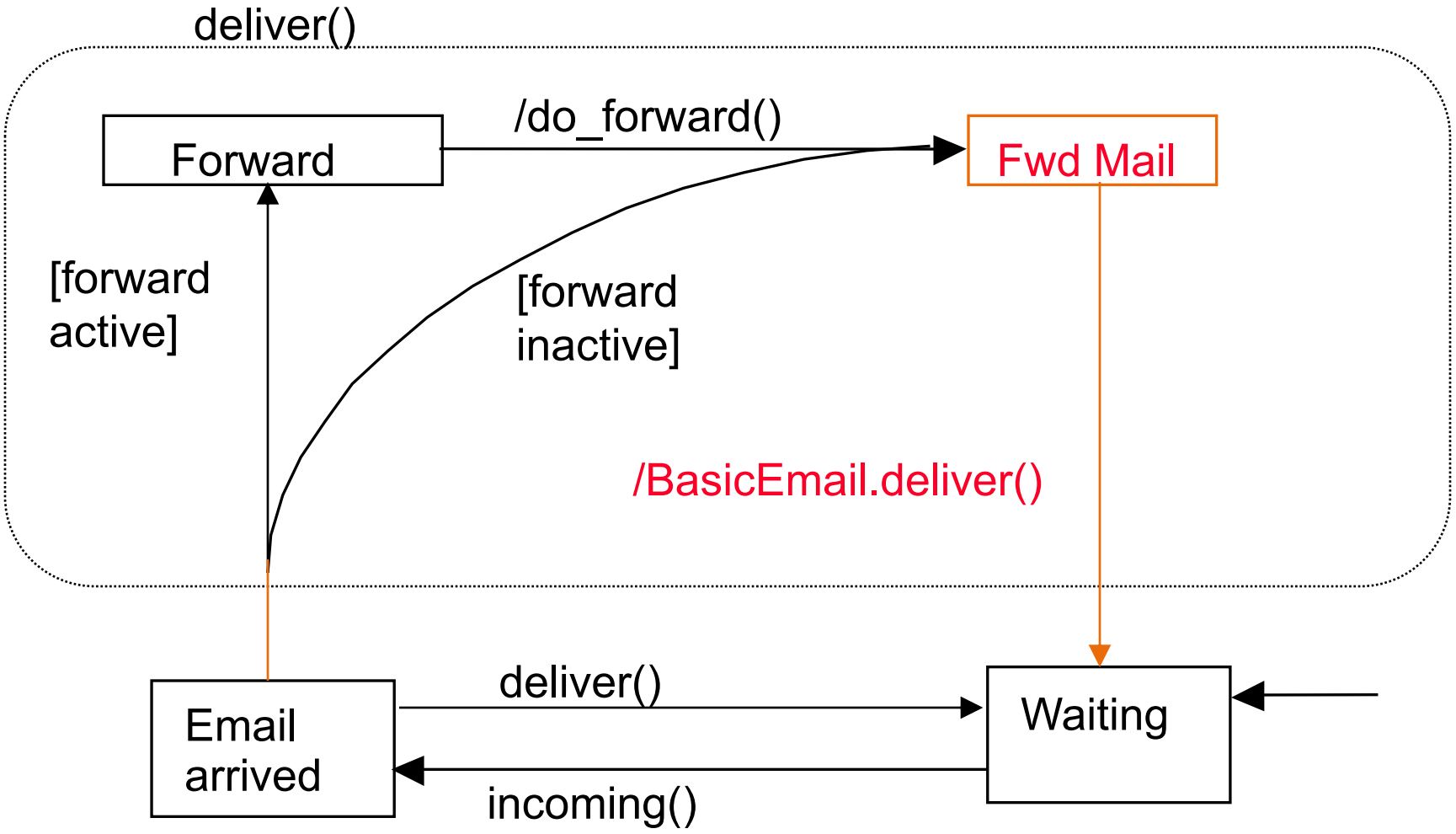
Combination by refinement of transitions

- Rules for adapting one feature to the other („interaction handling“)
  BasicEmail → Decryption/Forwarding
- Refinement describes the internal behavior of a transition by a statechart



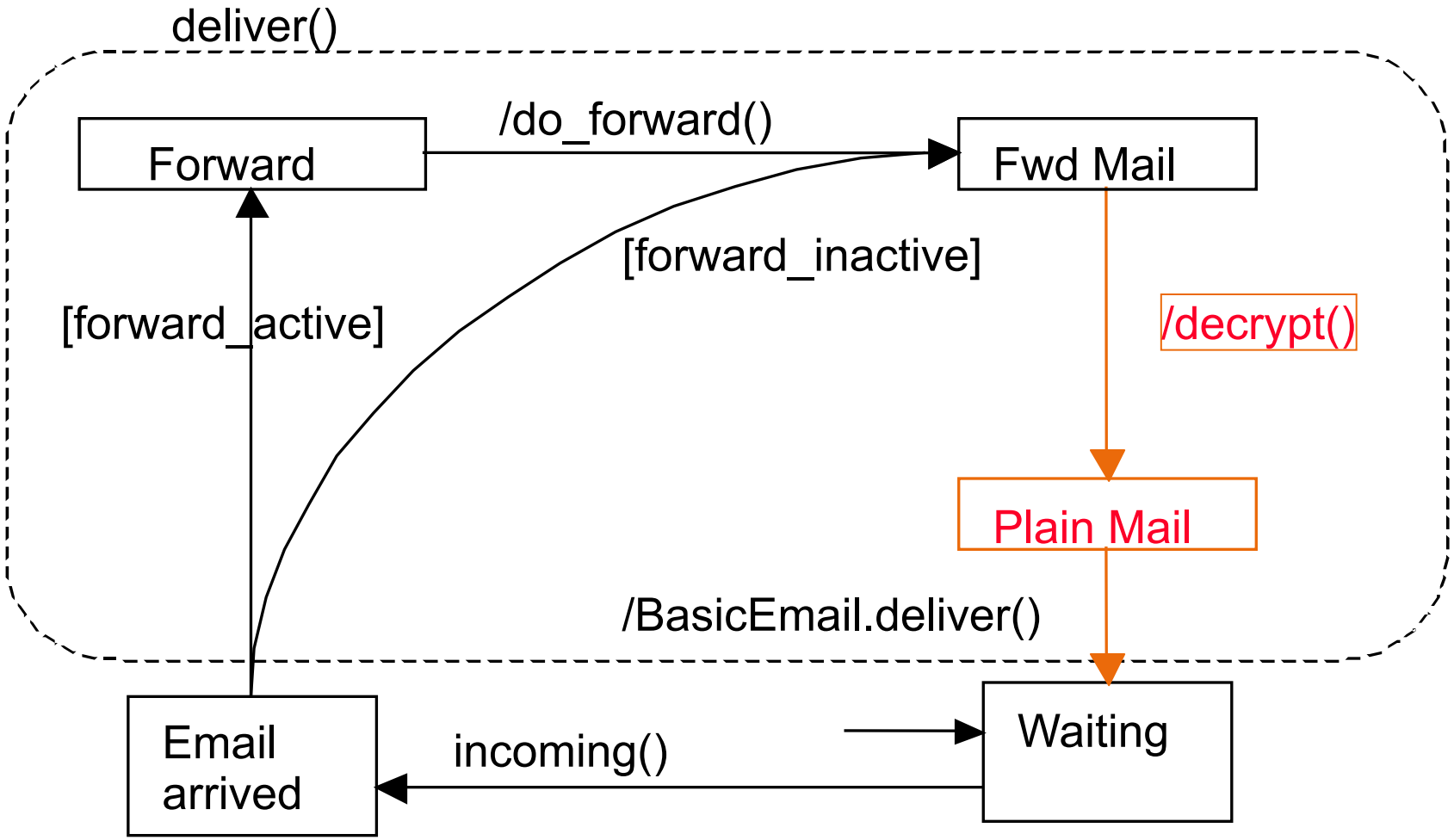
© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Transition Refinement: Basic Email and Forwarding



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Combination of 3 Features (1)
Basic Email, Forwarding and Decryption

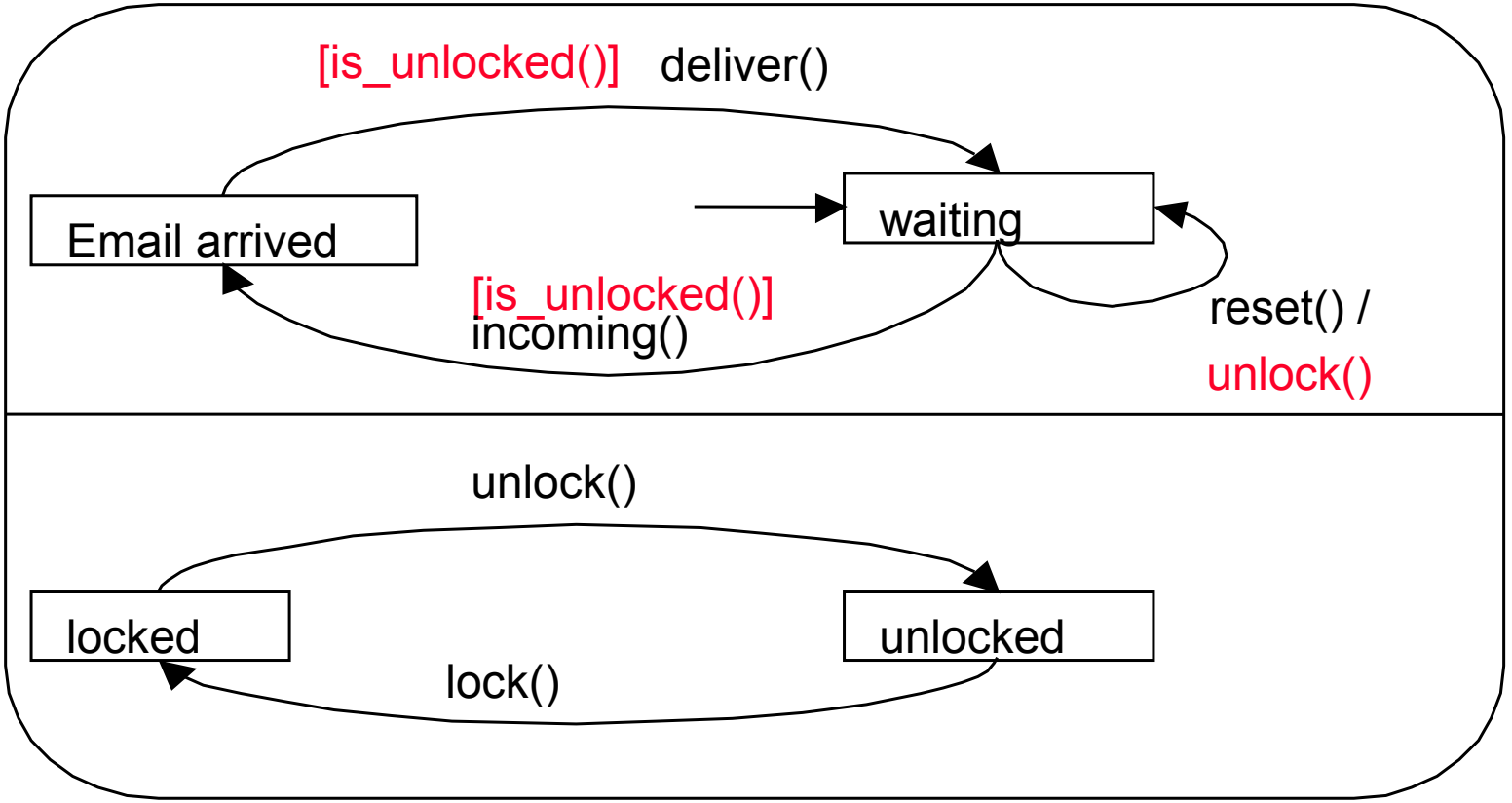


© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Combination Rules for Base Features

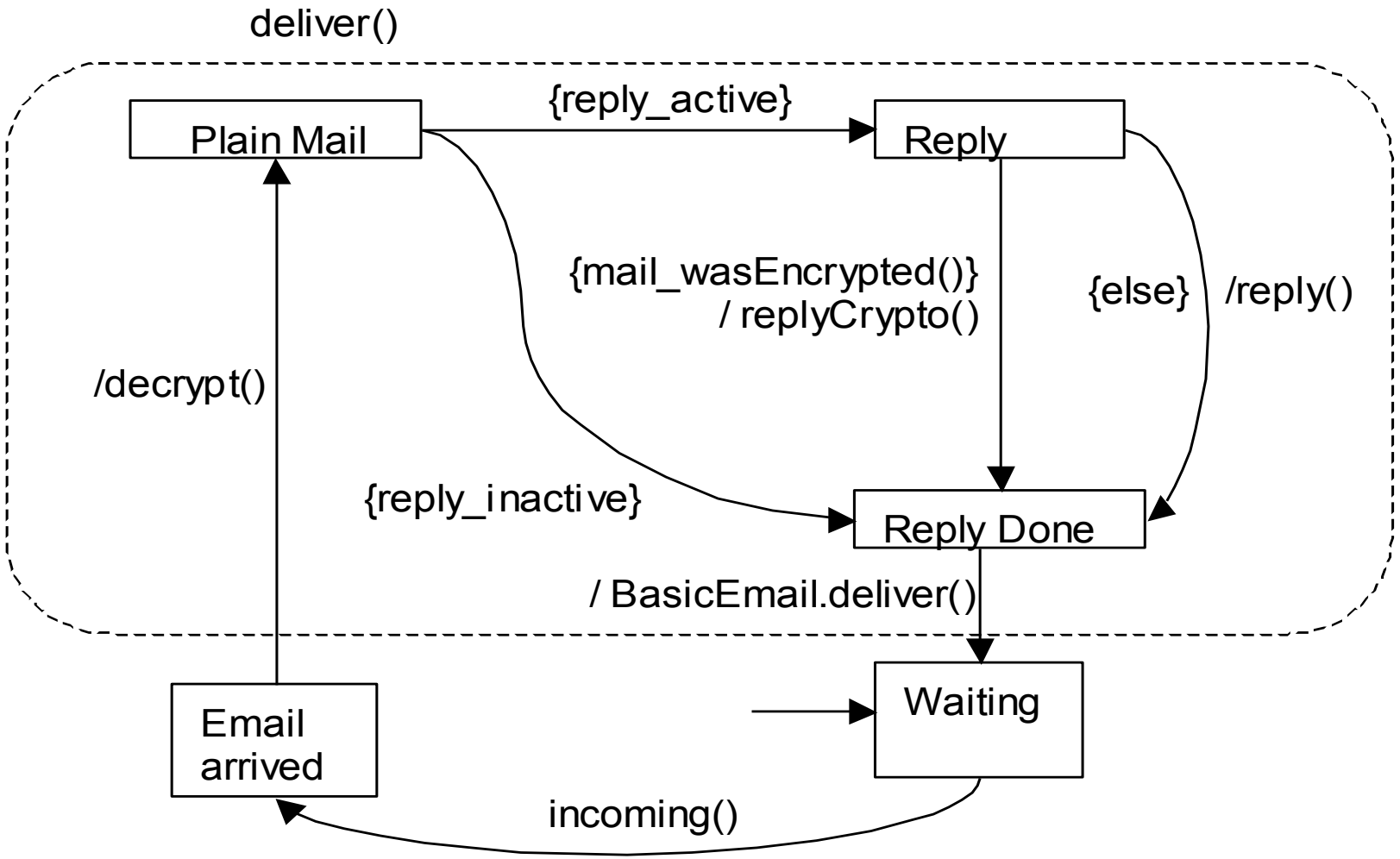
Combination by parallel composition

- Example: Lock-Feature with conditional refinement



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Combination of 3 Features (2): Basic Email, Auto Reply und Decryption



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Summary Feature Modeling

- Refinement on graphical models is possible (Prehofer, Sosym 2004)
- But: only handled simple state charts with specific semantics
  - Full UML much more complex
  - Above semantics different than FOP
- Need good graphical representation of interactions and features
  - Need more tools here
- Compare to recent work on Aspect-oriented modeling
  - E.g. Robert France, Gefei Zhang/LMU Munich, …
  - Challenges are complexity of UML and AOP semantics

© Fraunhofer ESK

Summary and Outlook

- FOP has been focusing on behavior and semantic composition
  - Modular design and composition as property-preserving refinement
- Features as a consistent model for requirements, design model, implementation and run time adaptation
- Future challenges
  - Features at run time – self-adaptation and control needed
  - Distributed features

© Fraunhofer ESK

Outlook:
Adaptive, distributed systems and distributed features

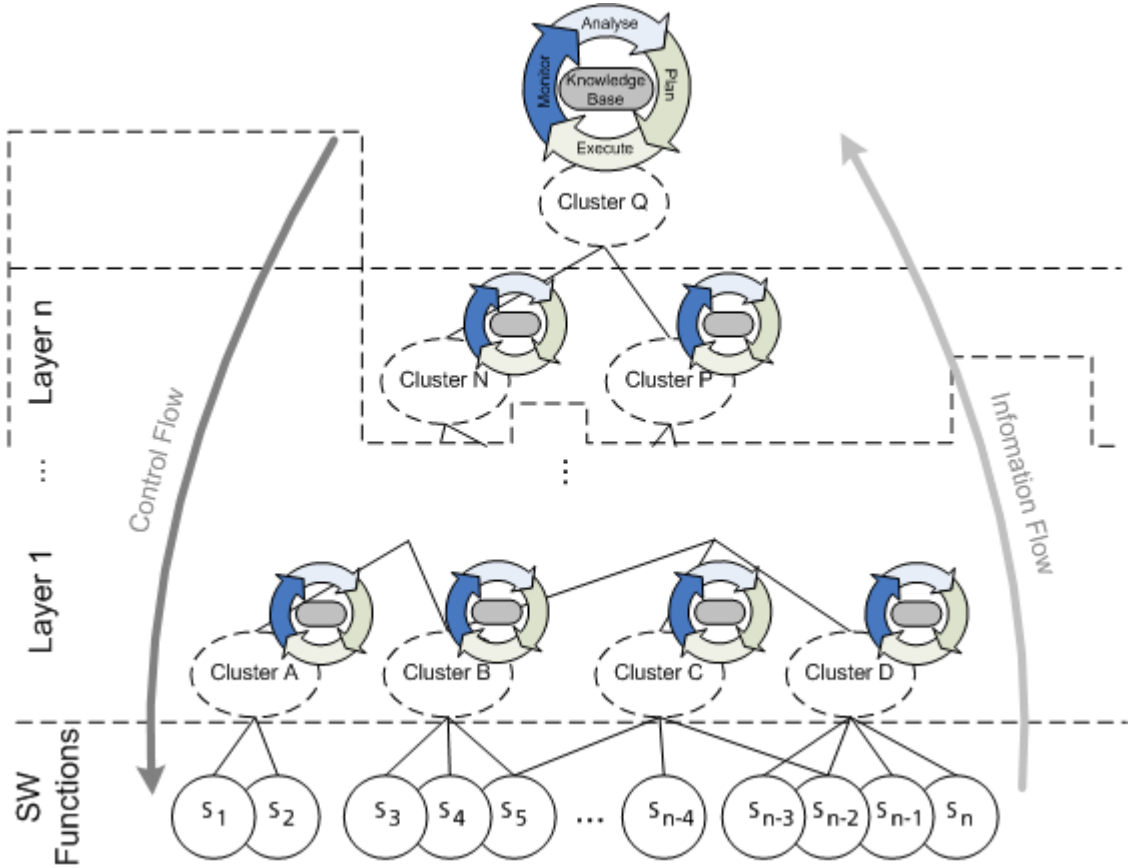
Example: Automotive System with 70 ECUs and distributed features

Research challenges

-Suitable architecture models
  - Here: Multi-layer

-Control & management
  - Here: MAPE-K from Autonomic Computing

-Distributed reconfiguration



© Fraunhofer ESK
www.esk.fraunhofer.de/EN

Questions ?