Konfigurierbare Systemsoftware (KSS)

VI 2 – Software Product Lines

Daniel Lohmann

Lehrstuhl für Informatik 4 Verteilte Systeme und Betriebssysteme

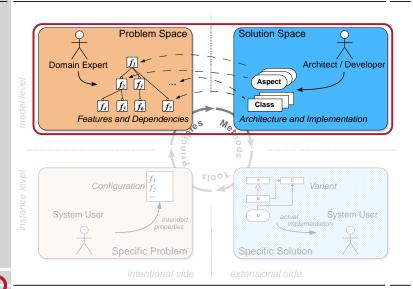
Friedrich-Alexander-Universität Erlangen-Nürnberg

SS 15 - 2015-04-27

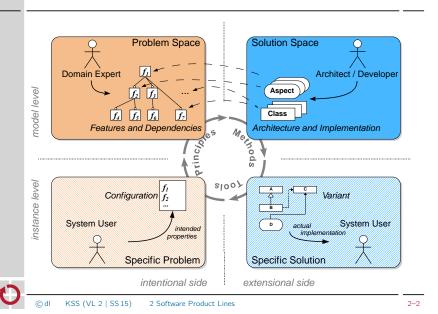
http://www4.informatik.uni-erlangen.de/Lehre/SS15/V_KSS



About this Lecture



About this Lecture



Agenda

- 2.1 Motivation: The Quest for Variety 2.2 Introduction: Software Product Lines
- 2.3 Case Study: i4Weathermon
- 2.4 Problem Space 2.5 Solution Space
- 2.6 References



Agenda

2.1 Motivation: The Quest for Variety Model Car Industry

Challenges



Model Car Industry: Variety Increase

- In the 1980s: little variety
 - Option to choose series and maybe a few extras (tape deck, roof rack)
 - A single variant (Audi 80, 1.3l, 55 PS) accounted for 40 percent of Audi's total revenue
- Twenty years later: built-to-order
 - 10²⁰ possible variants Audi:
 - BMW: 10³² possible variants
 - At average there are 1.1 equal instances of an Audi A8 on the street
- → **Product lines** with fully automated assembly



2 Software Product Lines | 2.1 Motivation: The Quest for Variety

Model Car Industry: Variety of an BMW X3



Roof interior: **90000** variants available

Car door: **3000** variants available *Unternehmensergebnis* **)**

324 variants available

66 Varianten sind ein wesentlicher Hebel für das

Franz Decker (BMW Group)

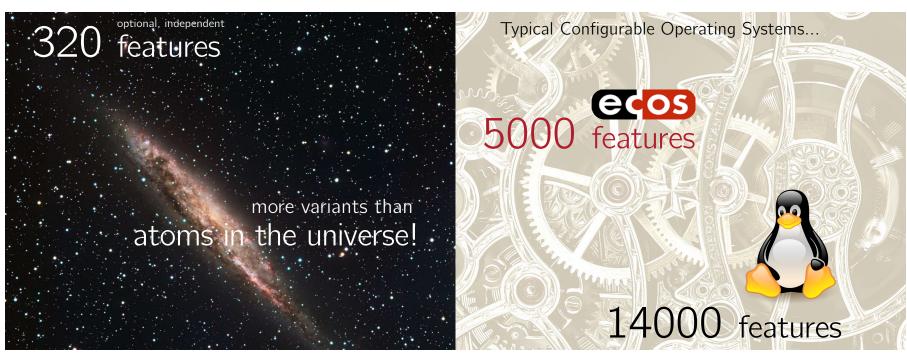
■ Rear axle:

2 Software Product Lines | 2.1 Motivation: The Quest for Variety

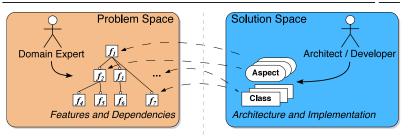
optional, independent features



one individual variant. for each human being



Challenges



- 1 How to identify the actually desired variability?
- 2 How to express the intended variability?
- **3** How to **implement** this variability in the code?
- 4 How to map variability options to the code?

Agenda

- 2.2 Introduction: Software Product Lines Terms and Definitions SPL Development Process Our Understanding of SPLs





Definition: (Software) Product Line, Feature

Product Line (Withey)

(Definition 1)

66 A **product line** is a group of products sharing a common, managed set of **features** that satisfy the specific needs of a selected **market**.

Withey 1996: Investment Analysis of Software Assets for Product Lines [12]

Software Product Line (SEI)

(Definition 2)

66 A **software product line (SPL)** is a set of software-intensive systems that share a common, managed set of **features** satisfying the specific needs of a particular **market** segment or mission and that are developed from a common set of core assets in a prescribed way.

Northrop and Clements 2001: Software Product Lines: Practices and Patterns [8]

Remarkable:

SPLs are not motivated by **technical** similarity of the products, but by **feature** similarity wrt a certain **market**



@dl KSS (VI 2 I SS 1

2 Software Product Lines | 2.2 Introduction: Software Product Lines

2-12

The Emperors New Clothes?

Program Family

(Definition 4)

66 Program families are defined [...] as sets of programs whose common properties are so extensive that it is advantageous to study the common properties of the programs before analyzing individual members. **97**

Parnas 1976: "On the Design and Development of Program Families" [10]

- Most research on operating-system *families* from the '70s would today qualify as work on software product lines [2, 4, 5, 9–11]
 - Program Family

 Software Product Line
- However, according to the definitions, the viewpoint is different
 - Program family: defined by similarity between programs → Solutions
 - SPL: defined by similarity between requirements
- → Problems
- ⇒ A program family implements a software product line
- In current literature, however, both terms are used synonymously
 - lacktriangleq Program Family \Longleftrightarrow Software Product Line

O-

Product Line (Withey)

(Definition 1)

((A product line is a group of products sharing a common, managed set of features that satisfy the specific needs of a selected market.)

Withey 1996: Investment Analysis of Software Assets for Product Lines [12]

Software Product Line (SEI)

(Definition 2)

66 A software product line (SPL) is a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.

Northrop and Clements 2001: Software Product Lines: Practices and Patterns [8]

Feature (Czarnecki / Eisenecker)

(Definition 3)

66 A distinguishable characteristic of a concept [...] that is relevant to some stakeholder of the concept. **22**

Czarnecki and Eisenecker 2000: Generative Programming. Methods, Tools and Applications [3, p. 38]



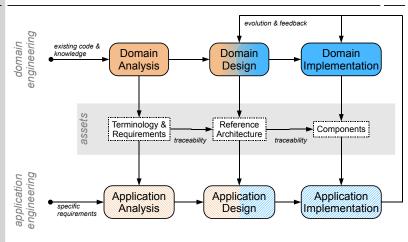
dl KSS (VL 2 | SS 1

2 Software Product Lines | 2.2 Introduction: Software Product Lines

2-12

SPL Development Reference Process

[1]



application engineering \mapsto **tailoring**



Our understanding: Configurable System Software

Configurability

(Definition 5)

Configurability is the property that denotes the degree of pre-defined variability and granularity offered by a piece of system software via an explicit **configuration interface**.

- Common configuration interfaces
 - Text-based: configure script or configure.h file (GNU tools)
 - configuration by commenting/uncommenting of (preprocessor) flags
 - no validation, no explicit notion of feature dependencies
 - Tool-based: KConfig (Linux, busybox, CiAO, ...), ecosConfig (eCos)
 - configuration by an interactive configuration editor
 - formal model of configuration space, hierarchical features
 - implicit/explicit validation of constraints



c) dl KSS (VL 2 | SS 15

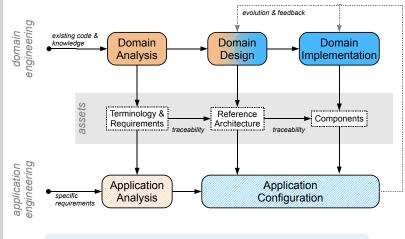
2 Software Product Lines | 2.2 Introduction: Software Product Lines

2-15

Agenda

- 2.1 Motivation: The Quest for Variety
- 2.2 Introduction: Software Product Lines
- 2.3 Case Study: i4Weathermon
- 2.4 Problem Space
- 2.5 Solution Space
- 2.6 References

Configurable SPL Reference Process



application engineering \mapsto configuring

0

dl KSS (VL 2 | SS 15

2 Software Product Lines | 2.2 Introduction: Software Product Lines

I²C

USB

2-16

The i4WeatherMon Weather Station



Display

- A typical embedded system
 - Several, optional sensors
 - Wind
 - Air Pressure
 - Temperature
 - Several, optional actuators (here: output devices)
 - LCD
 - PC via RS232
 - PC via USB
- To be implemented as a product line
 - Barometer: Pressure + Display
 - Thermometer: Temperature + Display
 - Deluxe: Temperature + Pressure+ Display + PC-Connection
- Outdoor: <as above> + Wind
- ...

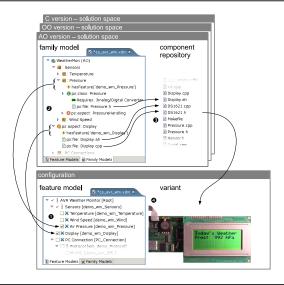


uController (AVR)



2-17

The i4WeatherMon Software Product Line





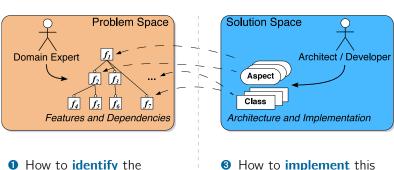
KSS (VL 2 | SS 15)

2 Software Product Lines | 2.3 Case Study: i4Weathermon

2-19

[7]

Challenges



- actually desired variability?
- **2** How to **express** the intended variability?
- **3** How to **implement** this variability in the code?
- 4 How to map variability options to the code?

Agenda

2.4 Problem Space Domain Analysis Feature Modelling



KSS (VL 2 | SS 15) 2 Software Product Lines | 2.4 Problem Space

2-20

Domain Analysis



Domain Scoping

- Selection and processing of domain knowledge
- Restriction of diversity and variety

Domain Modelling

- Systematic evaluation of the gained knowledge
- Development of a taxonomy

→ Domain Model

(Definition 6)

66 A **domain model** is an explicit representation of the **common** and the variable properties of the system in a domain, the semantics of the properties and domain concepts, and the dependencies between the variable properties. ??

> Czarnecki and Eisenecker 2000: Generative Programming. Methods, Tools and Applications [3]



Elements of the Domain Model

- Domain definition specifies the scope of the domain
 - Examples and counter examples
 - Rules for inclusion/exclusion of systems or features
- Domain glossary defines the vocabulary of the domain
 - Naming of features and concepts
- Concept models describe relevant concepts of the domain
 - Formal description (e.g., by UML diagrams)
 - Textual description
 - Syntax and semantics
- Feature models describe the common and variable properties of domain members
 - Textual description
 - Feature diagrams



2-23

KSS (VL 2 | SS 15) 2 Software Product Lines | 2.4 Problem Space

14WeatherMon: Domain Model (simplified)

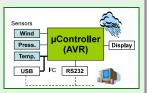
Domain Glossary: i4WeatherMon

- PC Connection: Optional communication channel to an external PC for the sake of continuous transmission of weather data. Internally also used for debug purposes.
- Sensor: Part (1 or more) of the i4WeatherMon hardware that measures a particular weather parameter (such as: temperature or air pressure).
- Actuator: Part (1 or more) of the i4WeaterMon hardware that processes weather data (such as: LCD).
- XML Protocol: XML-based data scheme for the transmission of arbitrary weather data over a PC Connection.
- SNG Protocol: Binary legacy data scheme for the transmission of wind, temperature and air pressure data only over a PC Connection. The data scheme is used by versions < 2.0 of PC Weather.

14WeatherMon: Domain Model (simplified)

Domain Definition: i4WeatherMon

■ The domain contains software for the depicted modular hardware platform. Future version should also support new sensor and actuator types (humidity, alarm, ...).



- The externally described application scenarios thermometer. PC. outdoor. ... shall be supported.
- The i4WeatherMon controller software is shipped in the flash memory of the μ C and shall not be changed after delivery.
- The i4WeatherMon shall be usable with all versions of the PC Weather client software.

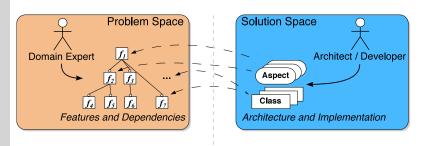
14WeatherMon: Domain Model (simplified)

Concept Models: i4WeatherMon

- XML Protocol: The following DTD specifies the format used for data transmission over a PC Connection:
 - <!ELEMEMENT weather ...> ...
- SNG Protocol: Wind, temperature and air pressure data are encoded into 4 bytes, sequentially transmitted as a 3-byte datagram over a PC Connection as follows:
- PC Connection ...



Challenges



- How to identify the actually desired variability?
- **2** How to **express** the intended variability?
- **3** How to **implement** this variability in the code?
- 4 How to map variability options to the code?



KSS (VL 2 | SS 15)

2 Software Product Lines | 2.4 Problem Space

2-27

[3]

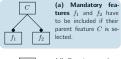
Feature Diagrams – Language

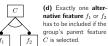
Syntactical Elements

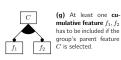
The filled dot • indicates a mandatory feature: $V = \{(C, f_1, f_2)\}\$



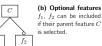
(a) Mandatory fea**tures** f_1 and f_2 have to be included if their parent feature C is selected.

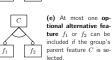
















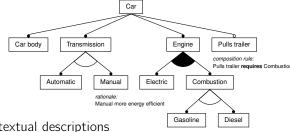




2-29

Feature Models

- Describe system variants by their commonalities and differences
 - Specify configurability in terms of optional and mandatory features
 - Intentional construct, independent from actual implementation
- Primary element is the **Feature Diagram**:
 - Concept (Root)
 - Features
 - Constraints



- Complemented by textual descriptions
 - Definition and rationale of each feature
 - Additional constraints, binding times, ...



KSS (VL 2 | SS 15)

2 Software Product Lines | 2.4 Problem Space

2-28

Feature Diagrams – Language

[3]

Syntactical Elements

A shallow dot o indicates an optional feature:

$$V = \{(C), (C, f_1), (C, f_2), (C, f_1, f_2)\}$$



(b) Optional features f_1 , f_2 can be included if their parent feature C is selected.



(a) Mandatory features f_1 and f_2 have to be included if their parent feature C is se-





native feature f1 or f2 has to be included if the group's parent feature C is selected.





(b) Optional features f_1 , f_2 can be included if their parent feature C is selected.







(h) Not used.



(c) Mandatory feature f_1 has to be included, optional feature f_2 can be included if their parent feature C



(f) Not used. Equivalent to (e).



(i) Not used. Equivalent to (h)

Feature Diagrams – Language

[3]

Feature Diagrams – Language

Syntactical Elements

a group of alternative

 $V = \{(C, f_1), (C, f_2)\}\$

 f_2 lected.

 f_2

 f_2

features:

The shallow arc △ depicts

(a) Mandatory fea

tures f_1 and f_2 have

to be included if their

parent feature C is se

(d) Exactly one alter-

native feature f_1 or f_2

has to be included if the

group's parent feature

(a) At least one cu-

mulative feature f_1, f_2

has to be included if the

group's parent feature

C is selected.

C is selected.

KSS (VL 2 | SS 15)

Syntactical Elements

[3]

(d) Exactly one alter-

native feature f_1 or f_2

has to be included if the

group's parent feature

(c) Mandatory fea

ture f_1 has to be in-

cluded, optional fea-

ture f_2 can be included

if their parent feature C is selected

(f) Not used

(i) Not used.

Equivalent to (b).

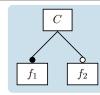
Equivalent to (e).

C is selected.

Syntactical Elements

Of course, both can be combined:

 $V = \{(C, f_1), (C, f_1, f_2)\}$



(c) Mandatory fea**ture** f_1 has to be included, optional fea**ture** f_2 can be included if their parent feature Cis selected.



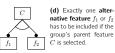




(b) Optional features f_1 , f_2 can be included if their parent feature C is selected.



(c) Mandatory feature f_1 has to be included, optional feature f_2 can be included if their parent feature C



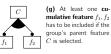


 f_2

(e) At most one optional alternative fea**ture** f_1 or f_2 can be included if the group's parent feature C is se-



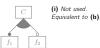
(f) Not used. Equivalent to (e).





Feature Diagrams – Language

(h) Not used. Eqivalent to (b).





KSS (VL 2 | SS 15)

2 Software Product Lines | 2.4 Problem Space

2-29

[3]

Feature Diagrams – Language

2-29 [3]

Syntactical Elements

The shallow arc △ depicts a group of alternative features:

 $V = \{(C), (C, f_1), (C, f_2)\}$



(e) At most one optional alternative fea**ture** f_1 or f_2 can be included if the group's parent feature C is selected.



(a) Mandatory features f_1 and f_2 have to be included if their parent feature C is selected.



(b) Optional features f_1 , f_2 can be included if their parent feature C is selected



(c) Mandatory feature f_1 has to be included, optional feature f_2 can be included if their parent feature C is selected



(d) Exactly one alternative feature f_1 or f_2 has to be included if the group's parent feature C is selected.



(e) At most one optional alternative feature f_1 or f_2 can be included if the group's parent feature C is se-



(f) Not used. Equivalent to (e)



group of cummulative features: $\mathcal{V} = \{(C, f_1), (C, f_2), (C, f_3), (C, f_4), (C, f_$ f_2), (C, f_1 , f_2)}



C

(b) Optional features

 f_1 , f_2 can be included

if their parent feature C

(e) At most one op-

tional alternative fea-

ture f_1 or f_2 can be

included if the group's

parent feature C is se-

(h) Not used.

2 Software Product Lines | 2.4 Problem Space

Egivalent to (b)

is selected.

(g) At least one cumulative feature f_1 , f_2 has to be included if the group's parent feature C is selected.



(a) Mandatory features f_1 and f_2 have to be included if their parent feature C is selected.

(d) Exactly one alter-

native feature f_1 or f_2

has to be included if the

group's parent feature

(g) At least one cu-

mulative feature f_1, f_2

has to be included if the group's parent feature C is selected.

C is selected.



(b) Optional features f_1 , f_2 can be included if their parent feature Cis selected.



(e) At most one optional alternative feature f_1 or f_2 can be included if the group's parent feature C is se-



(h) Not used. Enivalent to (h)



(c) Mandatory feature f_1 has to be included, optional feature f_2 can be included if their parent feature C



(i) Not used. Equivalent to (h)



(g) At least one cumulative feature f_1, f_2 has to be included if the group's parent feature C is selected



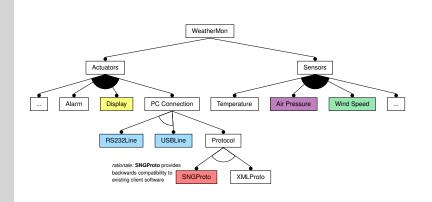


2-29



 f_2

14WeatherMon: Feature Model





2 Software Product Lines | 2.4 Problem Space

2-30

2 Software Product Lines | 2.5 Solution Space

Solution Space

Aspect

Architecture and Implementation

3 How to **implement** this

4 How to map variability options to the code?

variability in the code?

Class

2-31

Architect / Developer

Agenda

- 2.5 Solution Space

Reference Architecture

Implementation Techniques Overview

Variability Implementation with the C Preprocessor

Variability Implementation with OOP (C++)

Evaluation and Outlook



Challenges

Domain Expert

14WeatherMon: Reference Architecture

Functional decomposition (structure and process):

Problem Space

Features and Dependencies

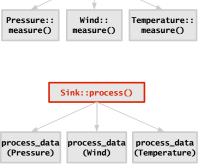
actually desired variability?

1 How to identify the

2 How to express the

intended variability?

```
int main() {
                                         Weather::measure()
  Weather data;
  Sink
        sink;
  while(true) {
                                Pressure::
                                 measure()
    // aquire data
    data.measure():
    // process data
    sink.process( data );
    wait();
                                (Pressure)
```



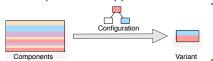




2-33

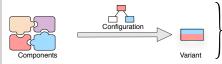
Implementation Techniques: Classification

Decompositional Approaches



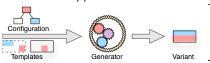
- Text-based filtering (untyped)
- Preprocessors

Compositional Approaches



- Language-based composition mechanisms (typed)
- OOP, AOP, Templates

Generative Approaches



- Metamodel-based generation of components (typed)
- MDD, C++ TMP, generators



© dl KSS (VL 2 | SS 15)

2 Software Product Lines | 2.5 Solution Space

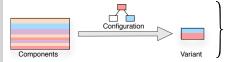
2-34

[6]

2-36

Implementation Techniques: The C Preprocessor

Decompositional Approaches



- Text-based filtering (untyped)
- Preprocessors (CPP)
- Conditional compilation with the C Preprocessor (CPP) is *the* standard approach to implement static configurability
 - Simplicity: the CPP "is just there"
 - Economy: CPP-usage does not involve any run-time overhead
 - Prominent especially in the domain of system software (Linux 3.2: 85000 #ifdef Blocks → "#ifdef hell")

Implementation Techniques: Goals

General

- Separation of concerns (SoC)
- 2 Resource thriftiness

Operational

- **6** Granularity Components should be fine-grained. Each artifact should either be mandatory or dedicated to a single feature only.
- **4** Economy

 The use of memory/run-time expensive language features should be avoided as far as possible. Decide and bind as much as possible at generation time.
- Pluggability Changing the set of optional features should not require modifications in any other part of the implementation. Feature implements should be able to "integrate themselves".
- **3** Extensibility The same should hold for new optional features, which may be available in a future version of the product line.

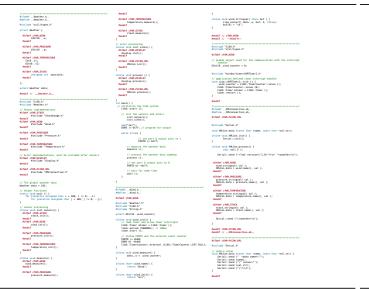


dl KSS (VL 2 | SS 15)

2 Software Product Lines | 2.5 Solution Space

2-35

I4WeatherMon (CPP): Implementation (Excerpt)





I4WeatherMon (CPP): Implementation (Excerpt)

```
struct Weather {
     #Ifdef chet,wise
#Untl6 .w;
#endif
                                                                                                                                                                                                                                                                     #ifdef cfWM_WIND
        Fifder chat.PRESSUR
UEntl6 _p;
Fendif
       Fifdef cfuPLT
IntR _t1;
UIntR _t2;
Fendif
                                                                                                                                                                                                                                                                     #ifdef cfWM_PRESSURE
                                                                                                                                                                                                                                                                                                        UInt16 _p:
        #Ifdef ch#LSTACK
unsigned int _maxstack;
feedif
                                                                                                                                                                                                                                                                     #endif
                                                                                                                                                                                                                                                                   #ifdef cfWM TEMPERATURE
                                                                                                                                                                                                                                                                               Int8 _t1;
   Finclude "CIAO.b"
Finclude "Weather.h"
                                                                                                                                                                                                                                                                              UInt8 _t2;
                                                                                                                                                                                                                                                                     #endif
                                                                                                                                                                                        // init the sensor
init_sensors()
init_sinks():
#Index of Mindex Principle "Wind.h"
                                                                                                                                                                                                                                                                   #ifdef cfWM_STACK
                                                                                                                                                                                    aum("emi");
00RD |= 8x7f; //
                                                                                                                                                                                                                                                                                                        unsigned int _maxstack;
     Pifdef cfWM_PRESSURE
Pinclude "Pressure.h"
                                                                                                                                                                                                             // set
PORTD |
                                                                                                                                                                                           // process th process ();
                                                                                                                                                                                                                                                                                                                                                       Finder chet.wish
wind stringval( val );
MMLCon.data ( wind.name(), val );
Meanif
                                                                                                                                                                                           // set port 0 out
PORTD &= -0x7f;
   #ifdef cfWLPCCDLXXL
#include "XMLConnection
feedif
   // The global weather data 
Weather data = (0);
   // helper functions
static void wait () {
   for (volatile unsigned char i = 100; i != 0; --i);
      for (volatile unsigned char j = 200; j != 0; --j);
                                                                                                                                                                                                                                                                                                                                                       #ifdef cfut_TEMPERATURE
  temperature_tringual( val );
  WLCon_data ( temperature_name(), val
                                                                                                                                                                           Finclude "GIAO.h"
Finclude "CIAO.h"
Finclude "String.h"
     // sensor processing inline wold init sensor side che-SHACK stack.init(); sendif sidef che.wood wind_init(); sendif
                                                                                                                                                                             extern UEnt16 _wind_counte
                                                                                                                                                                                                                                                                                                                                                           Serial::send ("</weather>\n");
                                                                                                                                                                                                                                                                                                                                                                                                                                             Sensor integration cross-
                                                                                                                                                                                                                                                                                                                                                  Pendif cfWM_PCCDM_XML
Pendif // _XMLConnection_ab_
                                                                                                                                                                                                                                                                                                                                                                                                                                                 cuts the central data
        #ifdef chm.TDMPERATURE
temperature_init();
fendif
                                                                                                                                                                                                                                                                                                                                                                                                                                                   structure, an interaction
                                                                                                                                                                                                                                                                                                                                                  // and a value of price that water water for the price price
                                                                                                                                                                               intine void wind measure() (
                                                                                                                                                                               inline char+ wind.name() {
                                                                                                                                                                           inline char* wind_unit() {
    return "m/s";
```

0

© dl KSS

KSS (VL 2 | SS 15) 2 Software Product Lines | 2.5 Solution Space

2

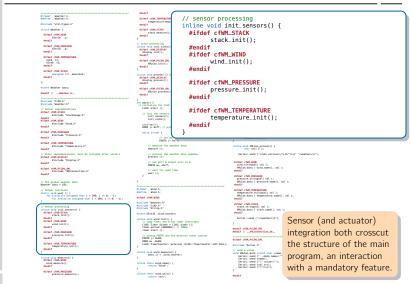
I4WeatherMon (CPP): Implementation (Excerpt)

```
inline void XMLCon_process() {
                                                                                                                                                                                                                                                                                                                                                      void wind_stringval( char* buf ) {
  itua_coevert( data._w, buf, 4, false);
  buf[4] = "\0";
                                     char val[ 5 ];
                Serial::send ("<?xml version=\"1.0\"?>\n" "<weather>\n");
                                                                                                                                                                                                                                                                                                                                          #include "CIAO.b"
#include "util/types.b"
                                                                                                                                                                                                                                                                                                                                           #1feef cfWLNIND
    #ifdef cfWM_WIND
             wind_stringval( val );
             XMLCon_data ( wind_name(), val );
                                                                                                                                                                                                                                                                                                                                           #include "hw/dev/timer/AVRTimer1.
                                                                                                                                                                                                                                                                                                                                             // application defined timer interrupt handler
void clasc:AMMTmerl::tick () {
.wind.counter = CLAD::TimerCounter::value ();
.CLAD::TimerCounter::value (0);
.CLAD::Timer Attack = CLAD::timer ();
.timer-restart ();
    #ifdef cfWM_PRESSURE
            pressure_stringval( val );
             XMLCon_data ( pressure_name(), val );
                                                                                                                                                                                                                                                                                                                                         #ifndef _XMLConnection_ah_
#define _XMLConnection_ah_
    #ifdef cfWM_TEMPERATURE
             temperature_stringval( val );
                                                                                                                                                                                                                                                                                                                                               nline void XMLCon_init() (
Serial::init();
              XMLCon_data ( temperature_name(), val );
    #ifdef cfWM_STACK
                                                                                                                                                                                                                                                                                                                                              #ifdef cfut_MIND
wind.stringual( val );
MMLCom_data ( wind_name(), val );
            stack_stringval( val );
XMLCon_data ( stack_name(), val );
                                                                                                                                                                                                                                                                                                                                              #ifdef cfmt,PRESSURE
pressure.stringval( val );
MLCon,data ( pressure_name(), val );
                                                                                                                                                                                                                                                                                                                                              #ifdef cfet_TEMPERATURE
temperature_stringual( val );
MLCom_data ( temperature_name(), val )
              Serial::send ("</weather>\n");
                                                                                                                                                                                                                                                                                                                                              #Sidef cdeM_STACK

stack stringual( val );

MMLCom_data ( stack_name(), val
                                                                                                                                                                                                                                                                                                                                               Serial::send ("</weather>\n");
                                                                                                                                                                                                                                                                                                                                                                                                                   Sensor integration also
                                                                                                                                                                                                                // load timer and allow timer interry
CLAD::Timer &timer = CLAD::Timer ();
timer.period (500000L); // 100mm
timer.start ();
                                                                                                                                                                                                                                                                                                                                                                                                                    crosscuts actuator code,
                                                                             pressure_init();
                                                                                                                                                                                                               // stetup PORTD
PORTD |= 6x80;
DORD Gm -0x80;
(140::Timer()---
                                                                                                                                                                                                                                                                                                                                                                                                                    an interaction between
                                                                                                                                                                                                                                                                                                                                          // some a value count that remains count that remai
                                                                                                                                                                                                           inline char- wind_name() {
    return "Wind";
                                                                                                                                                                                                           inline char* wind_unit() {
    return "m/m";
```

I4WeatherMon (CPP): Implementation (Excerpt)





© dl KSS (VL 2 | SS 15)

2 Software Product Lines | 2.5 Solution Space

2-37

(/)

I4WeaterMon (CPP): Evaluation

General

- Separation of concerns (SoC)
- 2 Resource thriftiness

Operational

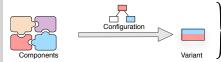
- Granularity
 - Components implement only the functionality of a single feature, but contain integration code for other optional features.
- 4 Economy
 - All features is bound at compile time.
- 6 Pluggability
 - Sensor integration crosscuts main program and actuator implementation.
- **6** Extensibility
 - New actuators require extension of main program.
 - New sensors require extension of main program and existing actuators.



2-37

Implementation Techniques: OOP

Compositional Approaches



- Language-based composition mechanisms (typed)
- OOP, AOP, Templates
- Object-oriented programming languages provide means for loose coupling by generalization and OO design patterns
 - Interfaces
 - → type substitutability (optional/alternative features)
 - Observer-Pattern
 - → quantification (cumulative feature groups)
 - Implicit code execution by global instance construction
 - → self integration (optional features)



© dl KSS (VL 2 | SS 15

2 Software Product Lines | 2.5 Solution Space

2-39

 (\mathbf{V})

1

str_val() init()

Pressure

measure()

registerSensor

Wind

measure()

name() unit()

str val()

init()

<u>id()</u>

unit()

id()

2-40

registerActuator

«alias»

PCConnection

SNGConnection

before_process()

Display

before_process()

process()

after_process()

process()

init()

#send()

I4WeaterMon (OOP): Evaluation

General

- Separation of concerns (SoC)
- 2 Resource thriftiness

Operational

- Granularity
 - Every component is either a base class or implements functionality of a single feature only.
- 4 Economy
 - Run-time binding and run-time type information is used
- only where necessary to achieve SoC.
- **6** Pluggability
 - Sensors and actuators integrate themselve by design patterns and global instance construction.
- 6 Extensibility
 - "Plug & Play" of sensor and actuator implementations.

0

2-41

© dl KSS (VL 2 | SS 15) 2 Software Product Lines | 2.5 Solution Space

I4WeatherMon (OOP): Design (Excerpt)

Sink

registerActuator()

«interface:

Actuator

before_process()

after_process()

process()

init()

(Colors reflect the respective features \hookrightarrow 2-30)

process()

Weather

registerSensor()

«interface»

Sensor

measure()

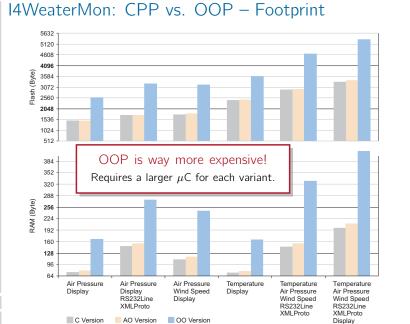
name()

str val()

init()

measure()

init()



tware Product Lines | 2.5 Solution Space

14WeaterMon: CPP vs. OOP - Footprint

| variant | version | text | data | bss | stack | = flash | = RAM | time (ms) |
|---------------------------|---------|------|------|-----|-------|---------|-------|-----------|
| Air Pressure, Display | С | 1392 | 30 | 7 | 34 | 1422 | 71 | 1.21 |
| | AO | 1430 | 30 | 10 | 38 | 1460 | 78 | 1.21 |
| | 00 | 2460 | 100 | 22 | 44 | 2560 | 166 | 1.29 |
| Air Pressure, Display, | С | 1578 | 104 | 7 | 34 | 1682 | 145 | 60.40 |
| RS232Line, XMLProto | AO | 1622 | 104 | 12 | 38 | 1726 | 154 | 59.20 |
| | 00 | 3008 | 206 | 26 | 44 | 3214 | 276 | 60.80 |
| Air Pressure, Wind Speed, | С | 1686 | 38 | 14 | 55 | 1724 | 107 | 2.96 |
| Display | AO | 1748 | 38 | 18 | 61 | 1786 | 117 | 2.96 |
| | 00 | 3020 | 146 | 33 | 65 | 3166 | 244 | 3.08 |
| Temperature, Display | С | 2378 | 28 | 8 | 34 | 2406 | 70 | 1.74 |
| | AO | 2416 | 28 | 11 | 38 | 2444 | 77 | 1.73 |
| | 00 | 3464 | 98 | 23 | 44 | 3562 | 165 | 1.82 |
| Temperature, Wind Speed, | С | 2804 | 90 | 17 | 35 | 2894 | 142 | 76.40 |
| Air Pressure, RS232Line, | AO | 2858 | 90 | 23 | 41 | 2948 | 154 | 76.40 |
| XMLProto | 00 | 4388 | 248 | 39 | 41 | 4636 | 328 | 76.40 |
| Temperature, Wind Speed, | С | 3148 | 122 | 17 | 57 | 3270 | 196 | 79.60 |
| Air Pressure, RS232Line, | AO | 3262 | 122 | 24 | 63 | 3384 | 209 | 77.60 |
| XMLProto, Display | 00 | 5008 | 300 | 44 | 67 | 5308 | 411 | 80.00 |



2 Software Product Lines | 2.5 Solution Space

2-43

Implementation Techniques: Summary

- CPP: minimal hardware costs but no separation of concerns
- OOP: separation of concerns but high hardware costs
- OOP cost drivers
 - Late binding of functions (virtual functions)
 - Calls cannot be inlined (→ memory overhead for small methods)
 - Virtual function tables
 - Compiler always generates constructors (for vtable initialization)
 - Dead code elimination less effective
 - Dvnamic data structures
 - Static instance construction
 - Generation of additional initialization
 - Generation of a global constructor to
 - Additional startup-code required

language concepts to achieve loose

Root of the problem:

→ AOP as an alternative.



With OOP we have to use dynamic coupling of static decisions.

CPP: minimal hardware costs – but no separation of concerns

OOP: separation of concerns – but high hardware costs

Implementation Techniques: Summary

- OOP cost drivers
 - Late binding of functions (virtual functions)
 - Calls cannot be inlined (→ memory overhead for small methods)
 - Virtual function tables
 - Compiler always generates constructors (for vtable initialization)
 - Dead code elimination less effective
 - Dvnamic data structures
 - Static instance construction
 - Generation of additional initialization functions
 - Generation of a global constructor table
 - Additional startup-code required

2 Software Product Lines | 2.5 Solution Space

Referenzen

- Günter Böckle, Peter Knauber, Klaus Pohl, et al. Software-Produktlinien: Methoden, Einführung und Praxis. Heidelberg: dpunkt.verlag GmbH, 2004. isbn: 3-80864-257-7.
- Fred Brooks. The Mythical Man Month. Addison-Wesley, 1975. isbn: 0-201-00650-2.
- [3] Krysztof Czarnecki and Ulrich W. Eisenecker. Generative Programming. Methods, Tools and Applications. Addison-Wesley, May 2000. isbn: 0-20-13097-77.
- Edsger Wybe Dijkstra. "The Structure of the THE-Multiprogramming System". In: Communications of the ACM 11.5 (May 1968), pp. 341–346.
- Arie Nicolaas Habermann, Lawrence Flon, and Lee W. Cooprider. "Modularization and Hierarchy in a Family of Operating Systems". In: Communications of the ACM 19.5 (1976), pp. 266-272.
- Jörg Liebig, Sven Apel, Christian Lengauer, et al. "An Analysis of the Variability in Forty Preprocessor-Based Software Product Lines". In: Proceedings of the 32nd International Conference on Software Engineering (ICSE '10). (Cape Town, South Africa). New York, NY, USA: ACM Press, 2010. doi: 10.1145/1806799.1806819.

Referenzen (Cont'd)

- [7] Daniel Lohmann, Olaf Spinczyk, and Wolfgang Schröder-Preikschat. "Lean and Efficient System Software Product Lines: Where Aspects Beat Objects". In: Transactions on AOSD II. Ed. by Awais Rashid and Mehmet Aksit. Lecture Notes in Computer Science 4242. Springer-Verlag, 2006, pp. 227–255. doi: 10.1007/11922827_8.
- [8] Linda Northrop and Paul Clements. *Software Product Lines: Practices and Patterns*. Addison-Wesley, 2001. isbn: 978-0-201-70332-0.
- [9] David Lorge Parnas. "On the Criteria to be used in Decomposing Systems into Modules". In: Communications of the ACM (Dec. 1972), pp. 1053–1058.
- [10] David Lorge Parnas. "On the Design and Development of Program Families". In: *IEEE Transactions on Software Engineering* SE-2.1 (Mar. 1976), pp. 1–9.
- [11] David Lorge Parnas. Some Hypothesis About the "Uses" Hierarchy for Operating Systems. Tech. rep. TH Darmstadt, Fachbereich Informatik, 1976.
- [12] James Withey. Investment Analysis of Software Assets for Product Lines. Tech. rep. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute, Nov. 1996.

