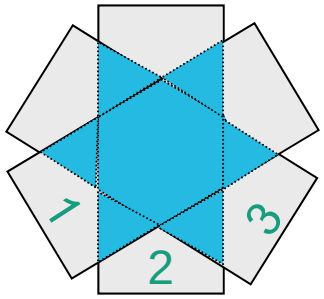
Product Line Engineering Lecture – PL Infrastructures II (5)

Dr. Martin Becker
martin.becker@iese.fraunhofer.de



Lectures - Schedule

Lectures

No	Date	Time	Location
1	21-Oct-11	15:30 - 17:00	48-462
2	28-Oct-11	15:30 - 17:00	IESE
3	4-Nov-11	15:30 - 17:00	IESE
4	11-Nov-11	15:30 - 17:00	IESE
5	18-Nov-11	15:30 - 17:00	48-462
6	25-Nov-11	15:30 - 17:00	48-462
7	2-Dec-11	15:30 - 17:00	IESE
8	9-Dec-11	15:30 - 17:00	IESE
9	16-Dec-11	15:30 - 17:00	IESE
10	6-Jan-12	15:30 - 17:00	IESE
11	13-Jan-12	15:30 - 17:00	IESE
12	20-Jan-12	15:30 - 17:00	IESE
13	27-Jan-12	15:30 - 17:00	IESE
14	3-Feb-12	15:30 - 17:00	IESE

Exercise Schedule

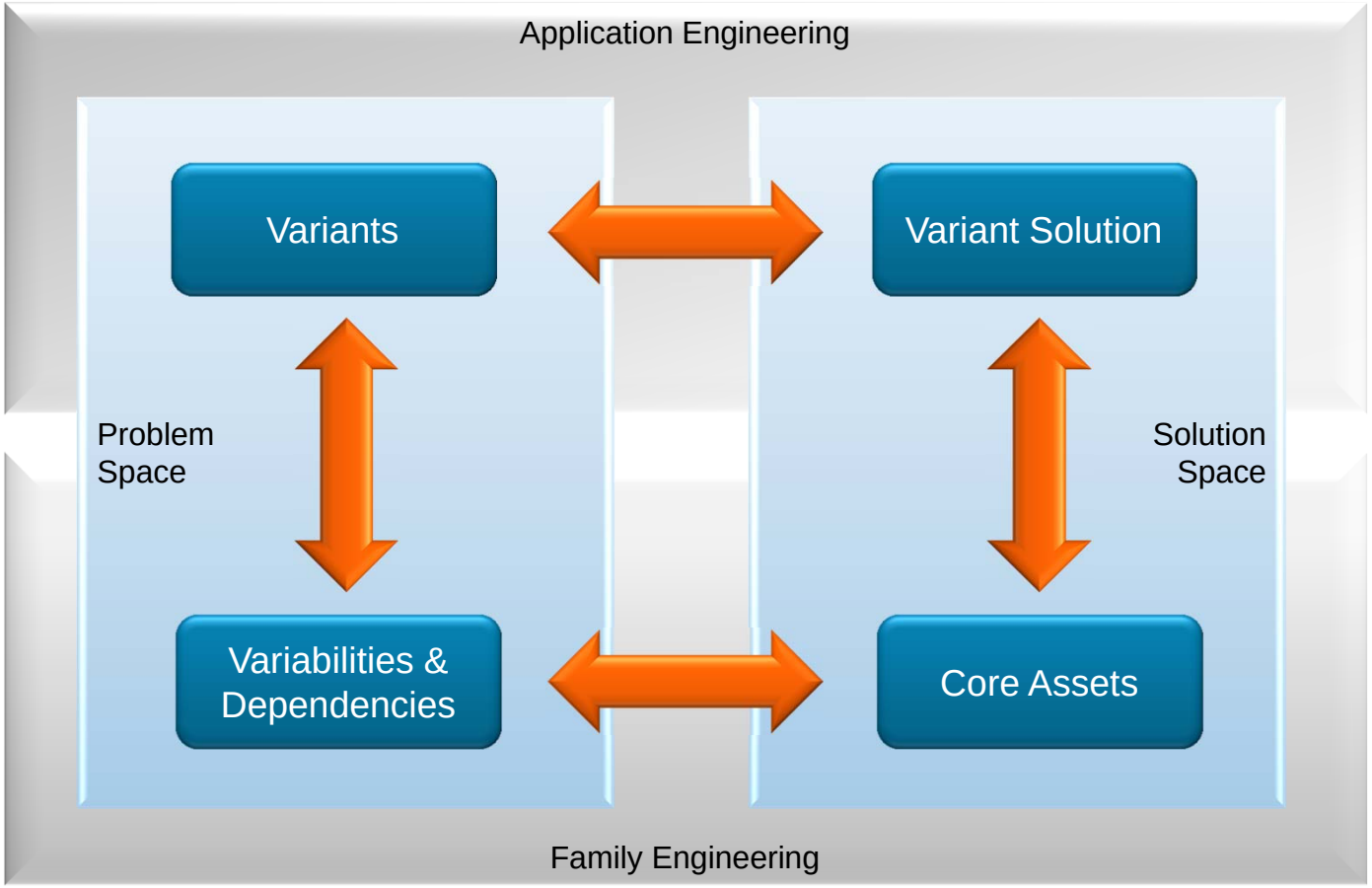
Exercises

No	Date	Time	Location
1	4-Nov-11	17:15 - 18:45	IESE
2	18-Nov-11	17:15 - 18:45	48-462
3	25-Nov-11	17:15 - 18:45	IESE
4	16-Dec-11	17:15 - 18:45	IESE
5	6-Jan-12	17:15 - 18:45	IESE
6	20-Jan-12	17:15 - 18:45	IESE
7	27-Jan-12	17:15 - 18:46	IESE
8	3-Feb-12	15:30 - 17:00	IESE

--- Product Line Infrastructure
Part II: Variability Realisation ---

How to realize
variability resolution support?

Separation of Concerns



Variability management interrelates the concerns

Fraunhofer IESE

Core Asset

Core Asset:==

asset that is

developed for reuse

in **more than one** product line member

Adapted from [Linden++07], [Metzger++07]

Core Assets

Content:

- Product Model, Process Model, Resource

Lifecycle Phase:

- Requirements, System Design, Unit Design,
Code, Image, Data, Test, Integration, Documentation, Configuration, Patch

Granularity:

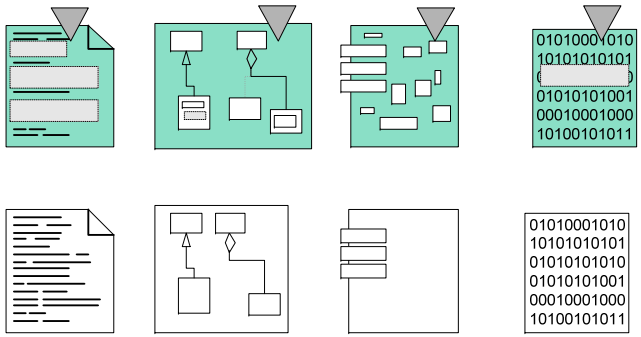
- Subsystem, Component, Folder,
Document, Document Fragment / Element

Genericity:

- Generic, Specific

Data Type:

- Model, Structured Text (e.g. XML), Text, Binary



What has to happen after the customer has selected his product?

Goal of a product line infrastructure is to facilitate the derivation of products, i.e the members of the product line

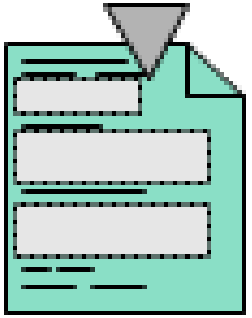
Variation Point

Variation Point :==

identifies **a location**

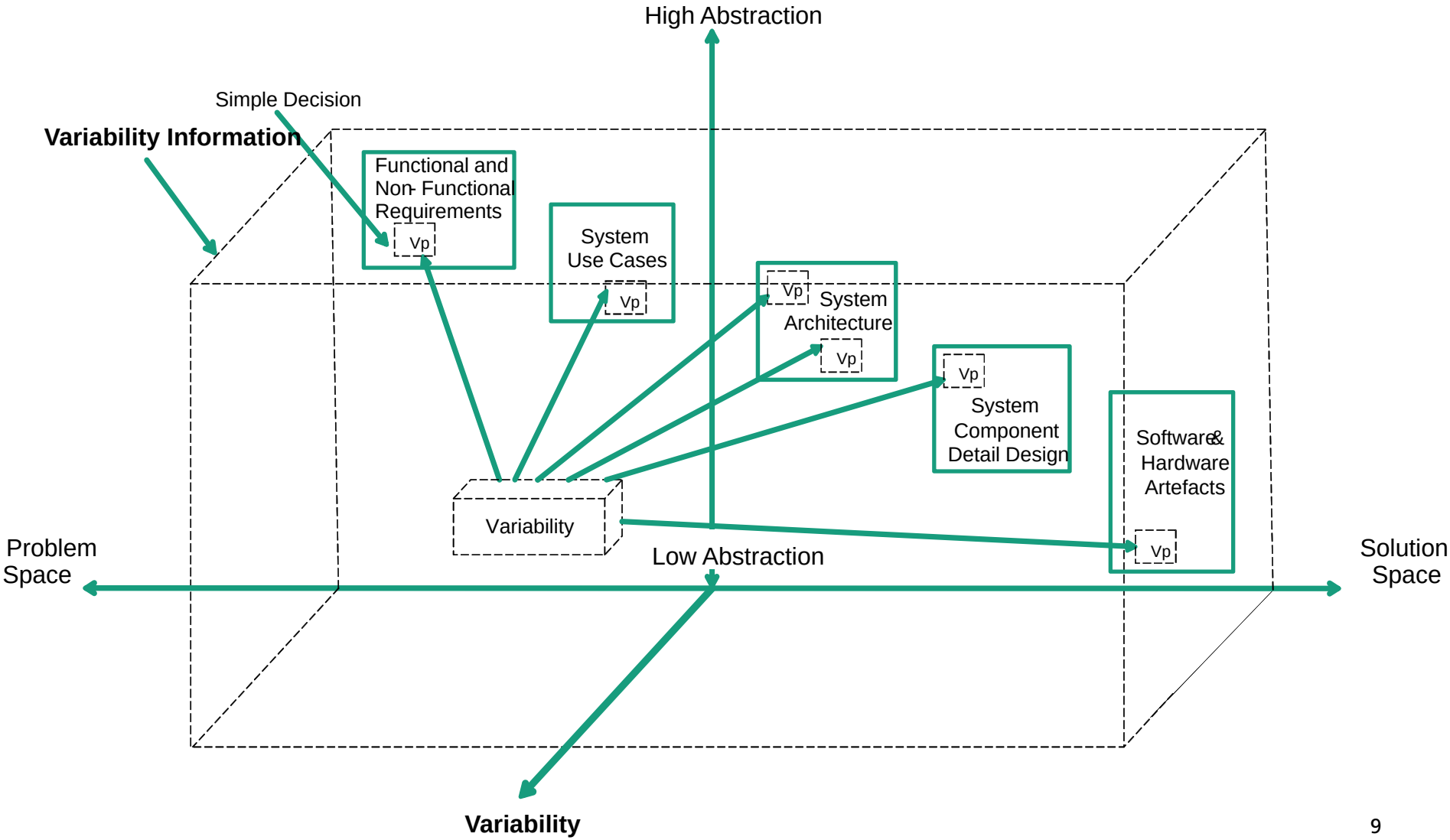
at which **variation** will occur

within **core assets**.



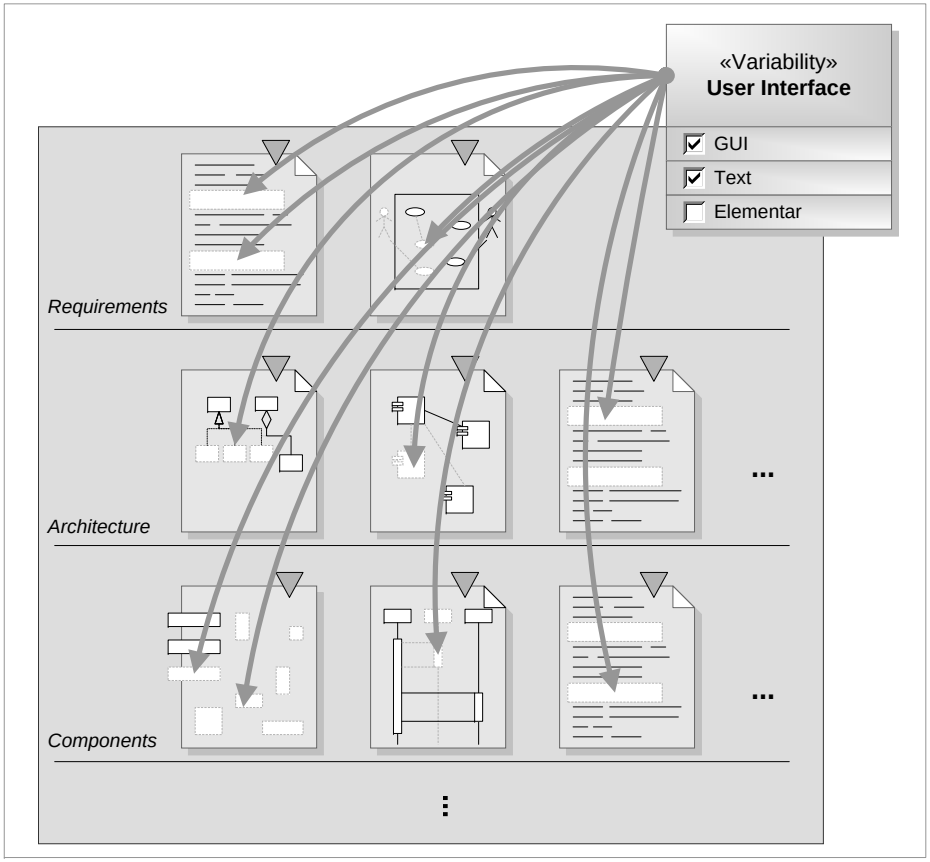
Goals: 1) to highlight where variant elements occur
(which makes variation easy to see and control);
2) to improve traceability of variability
(requires that goal 1 has been fulfilled).

Variability is a cross-cutting concern



Fraunhofer IESE

Example: Crosscutting Variability



Identification of Variation Points

Do not identify

Identify affected locations

Identify VP in Core Asset

- Markup, Tag, Stereotype

```
// sensor operations
// init_.. call before first use
// update_.. refeshes sensor value
VP_HAS_X_POS_SENSOR

«variant», «optional»
```

Unique identifier!

Identify VP externally

- List Points

```
Sensor.h, Line 35, Col5
…
Specification.mdl, UC2_34.Name
```

Fragility Problem

- Point Cut

```
before_execution (set*(*))
```

Understand Context

- Which variabilities do affect the VP?
- For which variant to provide what?
- What is the overall functionality?
- What is the current state?
- Which functions, data are available?
- Are there related VPs?


```
// clock abstraction
// clock value
extern int32_t the_clock;
// periodically set by ISR every sec
extern volatile bool period_elapsed;
// converts clock value to string
char* timetoa(int32_t);

/* ADD SENSOR VALUES HERE */

/* ADD SENSOR OERATIONS HERE */

bool event_happened=false;
int32_t event_time=0;
int16_t tilt_count=0;
int16_t tick=0;
```

Realise VP

- Enable instance specific adaption
  - ➔ Create an instance of the core asset
- Provision of realisation knowledge
- Provision of asset fragments
- Automated selection, generation, parameterisation

Provide appropriate realisation

Variability Mechanism

Variability Mechanism :==

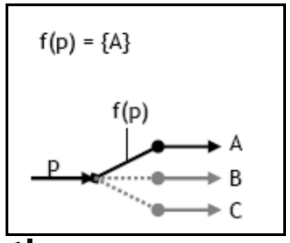
is a **particular way**

of **realizing variation**

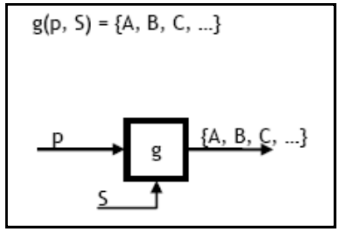
in **core assets**.

- Goals: 1) to efficiently package common & variant elements;
  2) to reduce evolution effort.

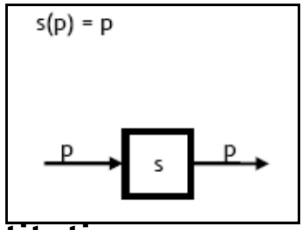
Variability Mechanism Primitives



selection



generation



substitution

- Selection
  - selecting predefined variants
  - e.g. component wiring, if-blocks, if-defs
- Generation
  - generating predefined variants
  - e.g. model-driven development
- Substitution
  - replacing a variation point by a value
  - e.g. parameterization
  - e.g. code weaving

Integration of Realisation Variant

Provided Realisation Variant needs to be added to the VP

Integrate the realisation variant into the Core Asset

Typically the VP is replaced by the Realisation Variant

- ➔ The VP is resolved
- ➔ After resolution of all VPs the Core Asset has been transformed into the specfic asset instance for the family member

Rebinding of Variability ?!?

Variant Management

Understand and manage the creation process of the instantiated application asset

- Separate core and variant (change)
- Understand the boundary of the VP
- Support diff and merge on the asset

Manage the core asset and the variants

General Purpose Approaches

- Templating
- Decision Modeling
- Preprocessing
  - CPP, M4, sed, scripting languages
  - Frame-Technology
  - Model-Editor automation
- Configuration Management

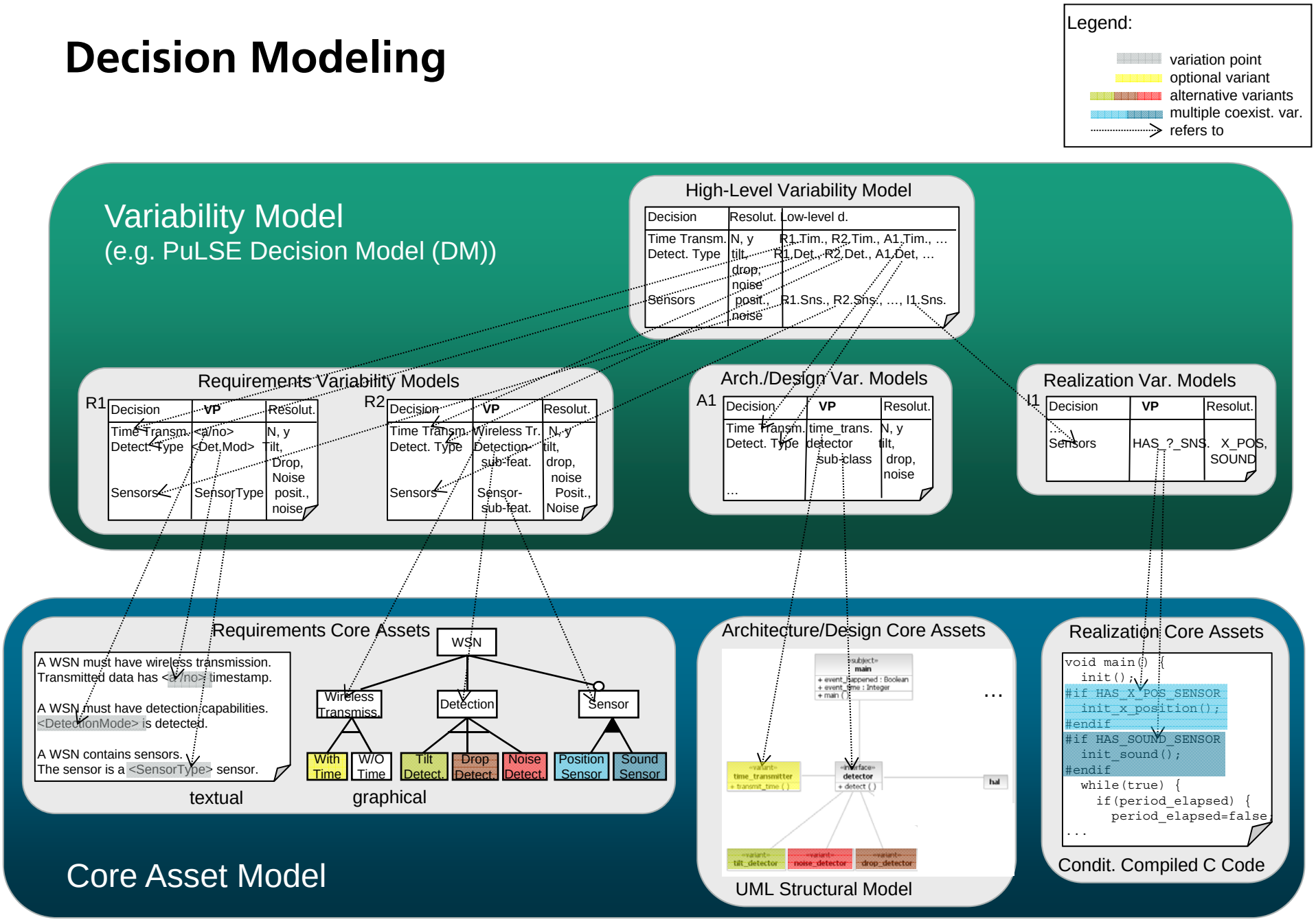
Templating (Form of cloning)

```
#include<string.h>
#include<stdio.h>
#include<stdbool.h>
#include<stdint.h>

// hardware initialization
void init();

// wireless transmission
// string to send
extern char send_buffer[61];
// sends send_buffer
void send();

// actuator abstractions
// switches led 2 on or off
void set_led_2(bool);
// toggles led 2: on <-> off
void toggle_led_2();

// clock abstraction
// clock value
extern int32_t the_clock;
// periodically set by ISR every sec
extern volatile bool period_elapsed;
// converts clock value to string
char* timetoa(int32_t);

/* ADD SENSOR VALUES HERE */
extern int16_t x_position;
/* END */

/* ADD SENSOR OERATIONS HERE */
// init_.. call before first use
// update_.. refeshes sensor value
void init_x_position();
void update_x_position();
/* END */

bool event_happened=false;
int32_t event_time=0;
int16_t tilt_count=0;
int16_t tick=0;
```

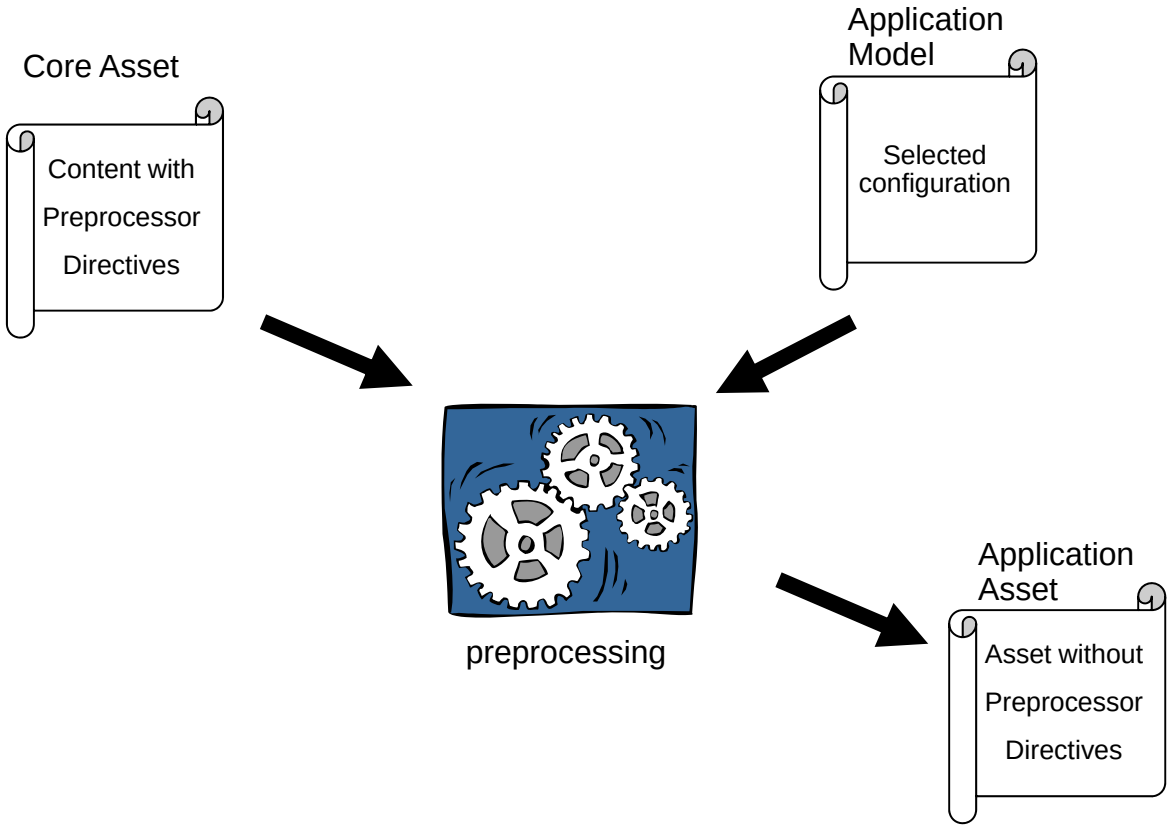
```
void main() {
  init();
/* ADD SENSOR INIT HERE */
init_x_position();
/* END */
  while(true) {
    if(period_elapsed) {
      period_elapsed=false;
/* ADD SENSOR UPDATE HERE */
    update_x_position();
/* END */
/* ADD DETECTION & TRANSMISSION HERE */
    if((x_position>(-100+25) && !event_happened)
     ||(x_position<(-100-25) && event_happened)) {
      event_happened=x_position>-100;
      if(event_happened) { // a tilt has started
        event_time=the_clock; // start one-shot timer
      }
      else { // a tilt has ended
        // has the device been tilted between 1 and 5s?
        if(the_clock-event_time>0
           && the_clock-event_time<=5) {
          toggle_led_2();
          tilt_count++;
        }
      }
    }
    tick=tick+1;
    if(tick%5==0) {
      tick=0;
      sprintf(send_buffer,"drink=%d",tilt_count*25);
/* END */
/* ADD PRE-TRANSMISSION BEHAVIOR HERE */
      if(event_happened) {
        strcat(send_buffer,",time=");
        strcat(send_buffer,
               timetoa(the_clock-event_time));
      }
/* END */
      send();
    }
  }
 }
}
```


Fraunhofer IESE

Decision Modeling



Overview Preprocessing



GPP ➔ Conditional Compilation

- Typically used in C/C++ environments
- Preprocessor
- Can be applied to any text / binary file

- Decouple common from variable code, so that the variable code is highlighted, and can be automatically included in or excluded from compilation.
- Conditional Compilation allows you to manage optional or alternative variable code next to common code, without adding new modules

GPP Commands

#define x y

This defines the user macro *x* as *y*

#if expr

allows evaluation of complex expressions: #if (VAR_X == A) || (VAR_X ==B)

#elif expr

can be used to avoid nested *#if* conditions: *#if* … *#elif* … *#endif*

#include file

open the specified file and evaluate its contents, inserting the resulting text in the current output

#exec command

execute external program and paste output ➔ External generators

Conditional Compilation: Example

```
#define T9_SUPPORTED
#undef  ATTACH_SUPPORTED
```

T9

Attachment

```
class Message {
public:
#ifdef T9_SUPPORTED
  void checkWordList() {...}
#endif
#ifdef ATTACH_SUPPORTED
  void enableAttachButton() {...}
#endif
};
class MessageUI {
public:
  void edit(Message &msg) {
#ifdef T9_SUPPORTED
    if(t9Active) tr.checkWordList();
#endif
    // perform editing
#ifdef ATTACH_SUPPORTED
    tr.enableAttachButton();
#endif
}};
```

One possible product Specification T9 supported

```
class Message {
public:
   void checkWordList() {...}
};
class MessageUI {
public:
  void edit(Message &msg) {
    if(t9Active)
      msg.checkWordList();
}};
```

Example: Conditional Compilation (CC) Code

(variant elements have the same colors as in the requirements on p.5)

```
line
 1 #include<string.h>
   #include<stdio.h>
   #include<stdbool.h>
   #include<stdint.h>
 5
   // hardware initialization
   void init();

   // wireless transmission
10 // string to send
   extern char send_buffer[61];
   // sends send_buffer
   void send();

15 // actuator abstractions
   // switches led 2 on or off
   void set_led_2(bool);
   // toggles led 2: on <-> off
   void toggle_led_2();
20
   // clock abstraction
   // clock value
   extern int32_t the_clock;
   // periodically set by ISR every sec
25 extern volatile bool period_elapsed;
   // converts clock value to string
   char* timetoa(int32_t);

   // sensor values
30 #if HAS_X_POS_SENSOR
   extern int16_t x_position;
   #endif
   #if HAS_SOUND_SENSOR
   extern int8_t sound;
35 #endif

   // sensor operations
   // init_.. call before first use
   // update_.. refeshes sensor value
40 #if HAS_X_POS_SENSOR
   void init_x_position();
   void update_x_position();
   #endif
   #if HAS_SOUND_SENSOR
45 void init_sound();
   void update_sound();
   #endif
```

```
49 bool event_happened=false;
50 int32_t event_time=0;
   int16_t tilt_count=0;
   int16_t tick=0;

   void main() {
55   init();
   #if HAS_X_POS_SENSOR
     init_x_position();
   #endif
   #if HAS_SOUND_SENSOR
60   init_sound();
   #endif
     while(true) {
       if(period_elapsed) {
         period_elapsed=false;
65 #if HAS_X_POS_SENSOR
       update_x_position();
   #endif
   #if HAS_SOUND_SENSOR
       update_sound();
70 #endif
   #if TILT_DETECTOR
       if((x_position>(-100+25) && !event_happened)
        ||(x_position<(-100-25) && event_happened)) {
         event_happened=x_position>-100;
75       if(event_happened) { // a tilt has started
           event_time=the_clock; // start one-shot timer
         }
         else { // a tilt has ended
           // has the device been tilted between 1 and 5s?
80         if(the_clock-event_time>0
              && the_clock-event_time<=5) {
             toggle_led_2();
             tilt_count++;
           }
85       }
       }
       tick=tick+1;
       if(tick%5==0) {
         tick=0;
90       sprintf(send_buffer,"drink=%d",tilt_count*25);
   #elif DROP_DETECTOR
       if((x_position>(-100+25) && !event_happened)
        ||(x_position<(-100-25) && event_happened)) {
         event_happened=x_position>-100;
95       // on change of tilt state, start a timer
         event_time=the_clock;
```

```
97       }
       if(event_time>0 && the_clock-event_time>=1)
       { // tilted >1s
100      set_led_2(event_happened);
         event_time=0;
         sprintf(send_buffer,"dropped=%d",
                 event_happened ?1 :0);
   #elif NOISE_DETECTOR
105    if(sound>20) {
         event_time=the_clock; //start one-shot t.
         event_happened=true;
       }

110    // forgetting
       if(the_clock-event_time>=10) {
         // forget when a presence was detected
         event_time=0;
         // forget about presence
115      event_happened=false;
       }
       set_led_2(event_happened);
       tick=tick+1;
       if(tick%5==0) {
120      tick=0;
         sprintf(send_buffer,"presence=%d",
                 event_happened ?1 :0);
   #endif
   #if HAS_TIME_TRANSMISSION
125      if(event_happened) {
           strcat(send_buffer,",time=");
           strcat(send_buffer,
                  timetoa(the_clock-event_time));
         }
130 #endif
         send();
       }
     }
   }
```


Fraunhofer IESE

Conditional Compilation: Advantages

- Variable parts are emphasized
  - Easy to see and control them
- Variable parts may crosscut the syntactical borders (function, class, or module boundaries) of the core asset language
- It can be introduced rapidly, because it is well-established, and it does not require additional tool support.

Conditional Compilation: Disadvantages

- Common parts and variable ones **reside together** mostly in the same module.
- It constrains application engineers to only selecting from among predefined variants, because preprocessor macros are **closed parameters**.
  - Extensions require changes in all parts that use a specific macro
- It is hard to ensure that the instantiated asset is always valid
  - Source code may become **incomprehensible** with large amount of conditional compilation macros.
- **Inconsistencies** in macro naming, macro usage, or macro configuration make it complicated to reuse source code across independently developed products.

Frame Technology

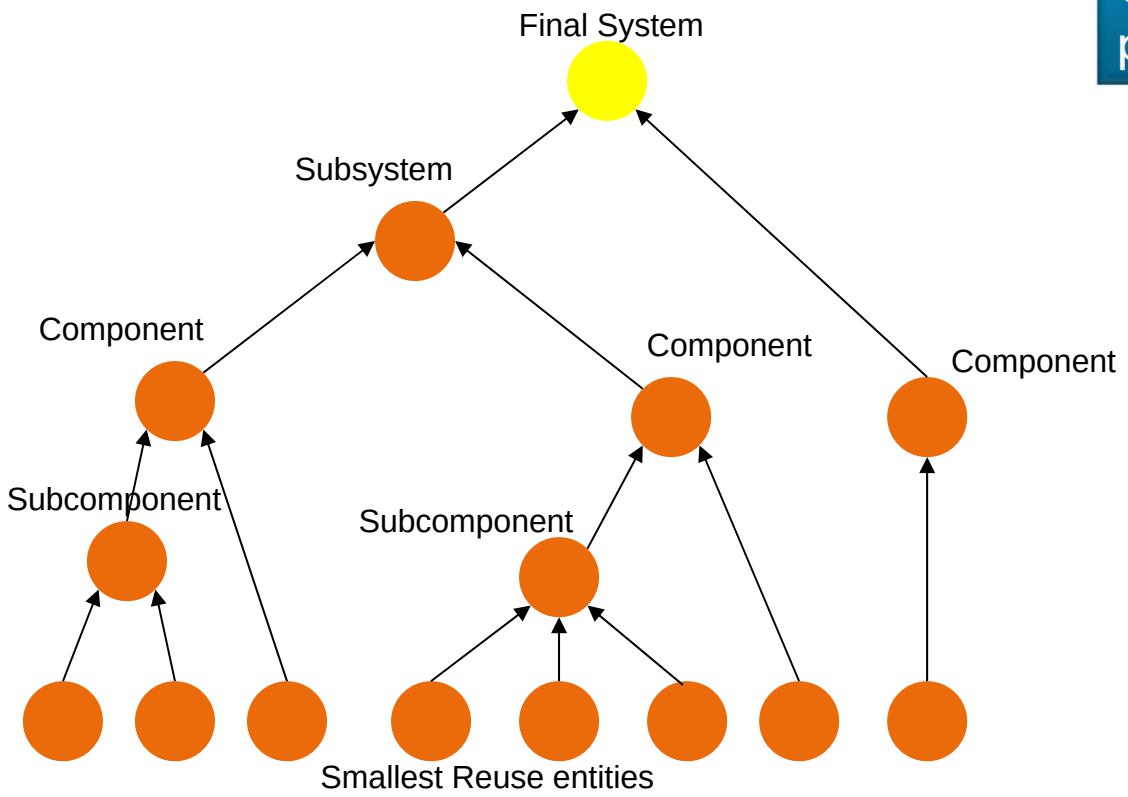
- **Decompose** textual information according to its **stability over time**
- Modules that need to change less frequently
  become nearly independent of modules that evolve more often.

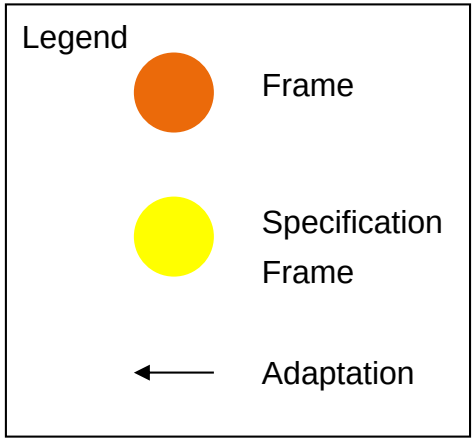
Frame Technology

- Characteristics
  - Same as conditional compilation except
    - More explicit variation points
    - Hierarchical scoping
    - Open parameters in addition to closed ones
- Examples:
  - XVCL
    - xml-based variant config. language
  - FrameProcessor

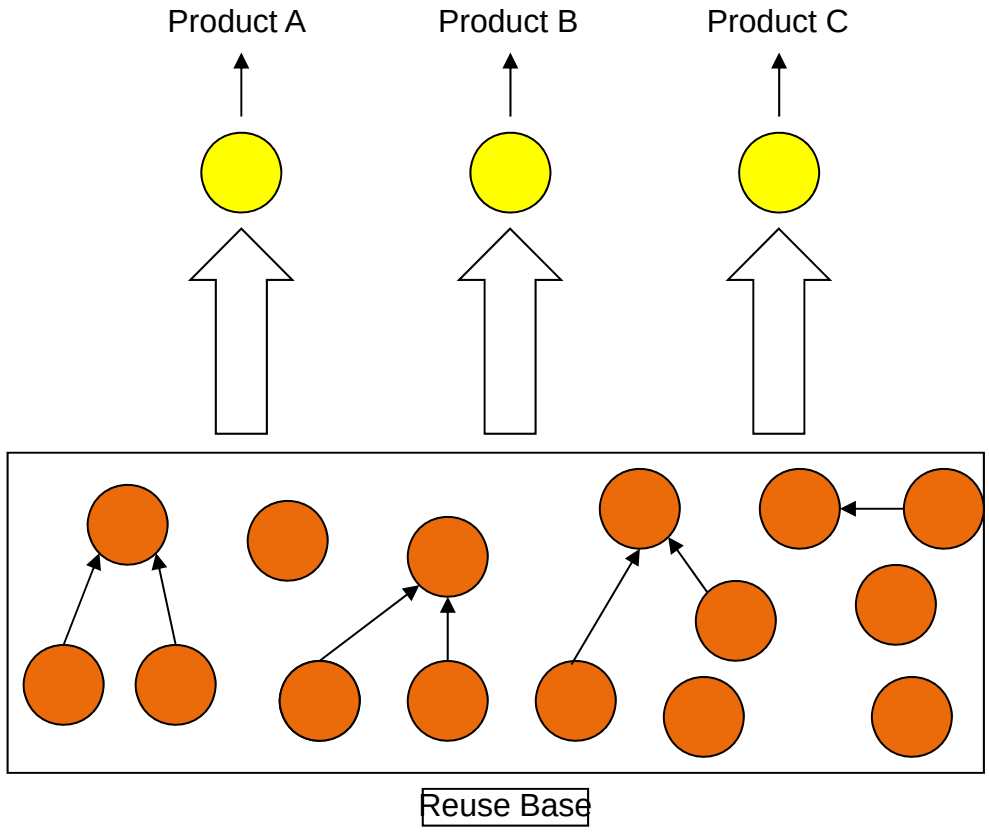
Frame Technology

Global overrides local principle supports reuse

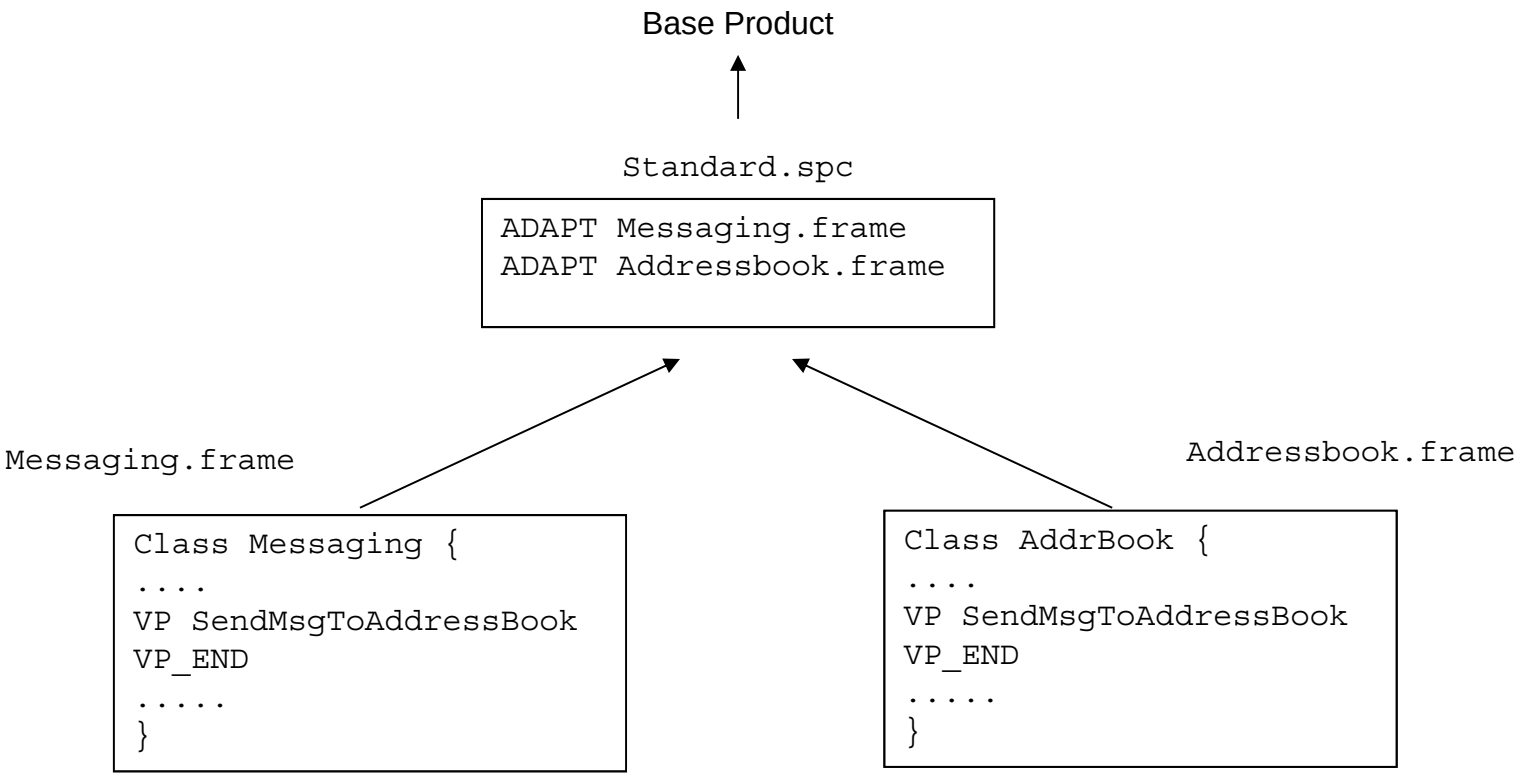




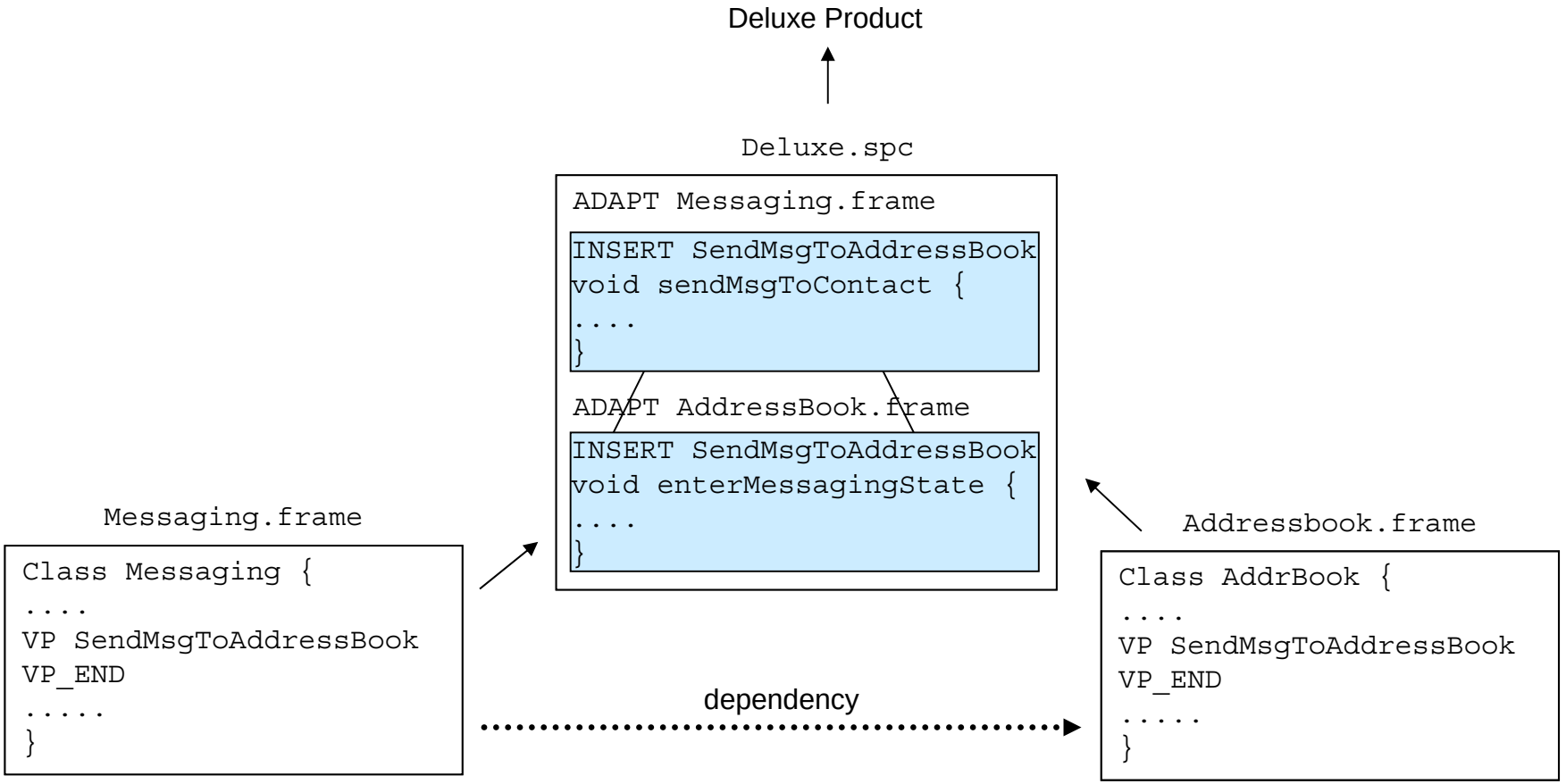
Frame Technology



Frame Technology



Frame Technology



Frame Technology (FT) Code (highlighted variant elements and VPs)

main:

```
outfile main.c
#include<string.h>
#include<stdio.h>
#include<stdbool.h>
#include<stdint.h>

// hardware initialization
void init();

// wireless transmission
// string to send
extern char send_buffer[61];
// sends send_buffer
void send();

// actuator abstractions
// switches led 2 on or off
void set_led_2(bool);
// toggles led 2: on <-> off
void toggle_led_2();

// clock abstraction
// clock value
extern int32_t the_clock;
// periodically set by ISR every sec.
extern volatile bool period_elapsed;
// converts clock value to string
char* timetoa(int32_t);

vp more_sensor_values
end

vp more_sensor_operations
end

bool event_happened=false;
int32_t event_time=0;
int16_t tilt_count=0;
int16_t tick=0;

void main() {
  init();
vp more_init
end
  while(true) {
    if(period_elapsed) {
      period_elapsed=false;
vp more_update
      if((x_position>(-100+25) && !event_happened)
       ||(x_position<(-100-25) && event_happened)) {
        event_happened=x_position>-100;
        if(event_happened) { // a tilt has started
          event_time=the_clock; // start one-shot timer
        }
        else { // a tilt has ended
          // has the device been tilted between 1 and 5s?
          if(the_clock-event_time>0
             && the_clock-event_time<=5) {
            toggle_led_2();
            tilt_count++;
          }
        }
      }
      tick=tick+1;
      if(tick%5==0) {
        tick=0;
        sprintf(send_buffer,
                "drink=%d",tilt_count*25);
end
        send();
      }
    }
  }
}
```

© Fraunhofer IESE

xpos_sensor:

```
adapt main
insert_after more_sensor_values
extern int16_t x_position;

insert_after more_sensor_operations
void init_x_position();
void update_x_position();

insert_after more_init
  init_x_position();

insert_before more_update
      update_x_position();
```

drop_detector:

```
adapt main
insert more_update
      if((x_position>(-100+25) && !event_happened)
       ||(x_position<(-100-25) && event_happened)) {
        event_happened=x_position>-100;
        // on change of tilt state, start a timer
        event_time=the_clock;
      }
      if(event_time>0 && the_clock-event_time>=1) {
        set_led_2(event_happened);
        event_time=0;
        sprintf(send_buffer,
                "dropped=%d",event_happened ?1 :0);
```

time_transmission:

```
adapt main
insert_after more_update
        if(event_happened) {
          strcat(send_buffer,",time=");
          strcat(send_buffer,
                 timetoa(the_clock-event_time));
        }
```

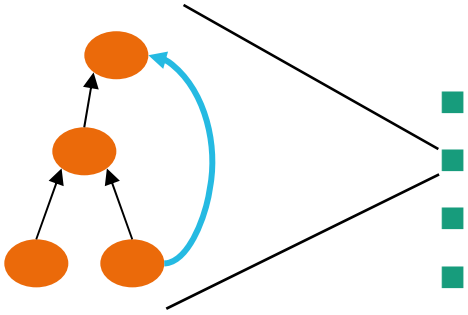
sound_sensor:

```
adapt main
insert_after more_sensor_values
extern int8_t sound;

insert_after more_sensor_operations
void init_sound();
void update_sound();

insert_after more_init
  init_sound();

insert_before more_elapsed
      update_sound();
```


Variation Point

Fraunhofer IESE

Frameprocessor (developed at Fraunhofer IESE) – Facts



- Comands
  - INSERT_BEFORE
  - INSERT_AFTER
  - INSERT
- User defined keywords
- N-level Adaptation
- Console application
- Open source (GPL)
  - Download at: http://sourceforge.net/projects/frameprocessor
  - Contact: Thomas Patzke <thomas.patzke@iese.fraunhofer.de>

Frame Technology: Advantages

- It allows arbitrary core assets to be managed as variabilities
  - even syntactically incomplete ones such as partial loops or isolated return statements.
- It facilitates variability management at explicit, different levels of context sensitivity, so that modules become nearly decomposable
- It has **no effects** on **resource efficiency**
- It **highlights variable parts**, and at the same time hides the common ones.
- Negative (contraction) and positive (extension) variabilities can be expressed
- It supports unpredicted changes, because variation points are **open parameters** and can be overridden in arbitrary ways.
- Adding new alternatives in alternative variabilities or multiple coexisting possibilities only leads to a linear growth in modules.

Frame Technology: Disadvantages

- Additional tool support and training are required.
- It cannot be used for developing black-box components whose implementation must always be hidden

Summary: Variability Realisation:
What has to happen after the customer has selected his product?

Identify affected locations	• Markup, List Points, Point Cuts
Understand context	• Provide background knowledge
Provide appropriate realisation	• Provision of asset fragments, automated selection, generation, provision of realisation knowledge
Integrate the realisation variant into the Core Asset	• Automated integration (inclusion) of parts
Manage the core asset and the variants	• Separate core and variant (change), support diff and merge

Further Reading

- Pohl, Klaus ; Böckle, Günter ; Linden, Frank van der: **Software Product Line Engineering : Foundations, Principles, and Techniques** Berlin : Springer-Verlag, 2005. - ISBN 3-540-24372-0
- Coplien, J. O.: **Multi-Paradigm Design for C++**. Addison-Wesley, 1998
- Bassett, P. G.: **Framing Software Reuse: Lessons from the Real World**. Prentice-Hall, 1997