Product Line Engineering Lecture – PL Infrastructures II (4)

Dr. Martin Becker
martin.becker@iese.fraunhofer.de
## Schedule - Lectures

<table>
<thead>
<tr>
<th>Date</th>
<th>Content</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>29-Oct-10</td>
<td>Introduction</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>5-Nov-10</td>
<td>Scoping</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>12-Nov-10</td>
<td>PL Infrastructure I (Variability Modelling)</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>19-Nov-10</td>
<td>PL Infrastructure II (Variability Realization)</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>26-Nov-10</td>
<td>no lecture</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>3-Dec-10</td>
<td>Configuration Management</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>10-Dec-10</td>
<td>PL Economics and Approaches</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>17-Dec-10</td>
<td>Requirements Engineering</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>7-Jan-11</td>
<td>PL-Architectures I</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>14-Jan-11</td>
<td>PL-Architectures II</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>21-Jan-11</td>
<td>Component Engineering</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>28-Jan-11</td>
<td>Quality Assurance</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>4-Feb-11</td>
<td>Organizational Issues / Adoption</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>11-Feb-11</td>
<td>Reengineering / Variant Analysis</td>
<td>15:30 - 17:00</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
</tbody>
</table>
Schedule - Exercises

<table>
<thead>
<tr>
<th>Date</th>
<th>Content</th>
<th>Time</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.11.2010</td>
<td>Scoping, Variability Modeling</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>03.12.2010</td>
<td>VM Realization</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>10.12.2010</td>
<td>Configuration Management</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>14.01.2011</td>
<td>PL Architectures</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>21.01.2011</td>
<td>Component Engineering</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
<tr>
<td>11.02.2011</td>
<td>Adoption, Variant Analysis</td>
<td>17:15 - 18:45</td>
<td>Z04.06 J. Nehmer (IESE)</td>
</tr>
</tbody>
</table>
--- Recap ---
Variability Modeling
Product Line Infrastructure

Product Line Infrastructure :=

- System of **facilities**, **equipment** and **services**
- needed for the **operation** of a product line

[specialization of ISO9000:2005-12]

Note: The product line infrastructure comprises the core asset base (aka. Product Line Artifact Base)
Goal of a product line infrastructure is to facilitate the derivation of products, i.e. the members of the product line.
Variability Management

- Variability management is a central component of every Product Line Engineering approach
- It allows managing common and varying parts as well as their interdependencies
  - Specification, Realisation
- It allows configuring, building and managing product line members
  - Production
On Variability

Variability := a characteristic that is different for some members of a category
[Bassett97]

Notes:

- Counterpart commonality: … same … all
- A kind of feature
- Delayed (design) decision: FE -> AE
  - Variability \(\rightarrow\) Decision
Recap: Variability Types

- **Optional Variability**
  - Boolean
  - Select 0..1 out of 1 possibility

- **Alternative Variability**
  - XOR
  - Enumeration
  - Select 1 out of n possibilities

- **Multiple Coexisting Possibilities**
  - OR
  - Set
  - Select 1..m out of n possibilities
Variability is a cross-cutting concern
Specifying / Modelling the variability

VARIATION POINTS – INTERNAL VIEW
⇒ positions in the artifact that can be adapted when the artifact is being reused
⇒ parameters, optional text blocks, place holders,…

VARIABILITY – EXTERNAL VIEW
⇒ “configurator” view on the artifact
⇒ information about dependencies between variations

Artifact

Variability Model

Decision Model
Feature Model
Configuration Model
How to realize variability resolution support?
Core Asset

Core Asset:==

asset that is

developed for reuse

in more than one product line member

Adapted from [Linden++07], [Metzger++07]
Product Model of Large SW Systems

[Source: http://wwwagse.informatik.uni-kl.de/teaching/gse/ws2010/docs/GSE_0_Overview_WS10.pdf]
Activities and Artefacts in the Product Life-cycle

[Source: http://vpe.mv.uni-kl.de/cms/index.php?id=314]
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
Variability Realisation:
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.

- Identify affected locations
- Understand context
- Provide appropriate realisation
- Integrate the realisation variant into the core asset
- Manage the core asset and the variants
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).
Identification of Variation Points

Do not identify

Identify VP in Core Asset

- Markup, Tag, Stereotype

  VP_HAS_X_POS_SENSOR «variant», «optional»

Identify VP externally

- List Points

  Sensor.h, Line 35, Col15
  Specification.mdl, UC2_34.Name

- 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?

```c
// 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 OPERATIONS 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 and 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**
  - 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

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 specific asset instance for the family member
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
General Purpose Approaches

- Templating
- Decision Modeling
- Preprocessing
  - CPP, M4, sed, scripting languages
  - Frame-Technology
  - Model-Editor automation
- Configuration Management
# Template (Form of cloning)

```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);

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

ADD SENSOR OPERATIONS HERE */
// init... call before first use
// update... refreshes 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++;
                }
            }
        } else {
            tick=tick+1;
            if(tick5==0) {
                tick=0;
                sprintf(send_buffer,"drink=\d",tilt_count*25);
            }
        }

        // ADD PRE-TRANSMISSION BEHAVIOR HERE */
        if(event_happened) {
            strcat(send_buffer,"",time="");
           畅销(send_buffer, timetoa(the_clock-event_time));
        }
        /* END */
        send();
    }
}
```
A WSN must have wireless transmission. Transmitted data has a timestamp.

A WSN must have detection capabilities. "Detection Mode" is detected.

A WSN contains sensors. The sensor is a "Sensor Type" sensor.
A decision model is a model that captures *variability* in a product line in terms of *open decisions* and *possible resolutions*.

<table>
<thead>
<tr>
<th>ID</th>
<th>Question</th>
<th>Variation Point</th>
<th>Resolution</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Can cups be warmed up?</td>
<td>coffee machine</td>
<td>yes</td>
<td>A warming plate is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>No warming plate is attached</td>
</tr>
<tr>
<td>2</td>
<td>Does the machine has a crushing mill for coffee, a slot for putting in already crushed coffee or both?</td>
<td>coffee machine</td>
<td>crushing mill</td>
<td>A crushing mill is installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>coffee slot</td>
<td>A slot for coffee powder is installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>both</td>
<td>A crushin mill and a slot for coffee powder is installed</td>
</tr>
<tr>
<td>3</td>
<td>Does the machine has a milk frother or a cappuccinatore?</td>
<td>coffee machine</td>
<td>milk frother</td>
<td>A milk frother is attached</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cappuccinatore</td>
<td>A cappuccinatore is attached</td>
</tr>
<tr>
<td>4</td>
<td>Does the machine have a container for coffee beans?</td>
<td>coffee machine</td>
<td>yes</td>
<td>A container for coffee beans is installed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>no</td>
<td>No container for coffee beans is installed</td>
</tr>
</tbody>
</table>

Decision models can be used *regardless* the type of the artifact (documents, models, code) => *orthogonal variability modeling*.
Overview Preprocessing

Core Asset
- Content with Preprocessor Directives

Application Model
- Selected configuration

Application Asset
- Asset without Preprocessor Directives

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
Conditional Compilation: Example

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

One possible product
Specification
T9 supported

```cpp
#define T9_SUPPORTED
#undef ATTACH_SUPPORTED
```
Example: Conditional Compilation (CC) Code  
(variant elements have the same colors as in the requirements on p.5)
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.
GPP- More Sophisticated 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
Frame Technology: Intent

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

- Frame Technology makes it easier to core assets localized that shares the same change rate, especially in cases where the syntax of the programming language would otherwise enforce this code to crosscut several modules.
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

Legend
- Frame
- Specification
- Adaptation
Frame Technology

Product A

Product B

Product C

Reuse Base
Frame Technology

Base Product

Standard.spc

ADAPT Messaging.frame
ADAPT Addressbook.frame

Messaging.frame

Class Messaging {
    ....
    VP_SendMsgToAddressBook
    VP_END
    ..... 
}

Addressbook.frame

Class AddrBook {
    ....
    VP_SendMsgToAddressBook
    VP_END
    ..... 
}
Frame Technology

Deluxe Product

Deluxe.spc

ADAPT Messaging.frame

INSERT SendMsgToAddressBook
void sendMsgToContact {
....
}

ADAPT AddressBook.frame

INSERT SendMsgToAddressBook
void enterMessagingState {
....
}

Class Messaging {
....
VP SendMsgToAddressBook
VP_END
....
}

Class AddrBook {
....
VP SendMsgToAddressBook
VP_END
....
}
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);

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

void main() {
  init();
  while(true) {
    if(period_elapsed) {
      period_elapsed=false;
    }
    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,
            "dropped=%d",tilt_count*25);
        send();
      }
    }
  }
}
```

adapt main
```
insert more_sensor_values
extern int16_t x_position;
insert more_sensor_operations
void init_x_position();
void update_x_position();
insert more_init
init_x_position();
insert more_update
update_x_position();
```

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 after more_update
update_sound();
```

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

---

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

---

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

```
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](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?

<table>
<thead>
<tr>
<th>Step</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identify affected locations</td>
<td>• Markup, List Points, Point Cuts</td>
</tr>
<tr>
<td>Understand context</td>
<td>• Provide background knowledge</td>
</tr>
<tr>
<td>Provide appropriate realisation</td>
<td>• Provision of asset fragments, automated selection, generation, provision of realisation knowledge</td>
</tr>
<tr>
<td>Integrate the realisation variant into the Core Asset</td>
<td>• Automated integration (inclusion) of parts</td>
</tr>
<tr>
<td>Manage the core asset and the variants</td>
<td>• Separate core and variant (change), support diff and merge</td>
</tr>
</tbody>
</table>
Further Reading