

Betriebssystemtechnik

Operating System Engineering (OSE)

Trennung der Belange bei der statischen Konfigurierung



1

Ergebnis der Problemanalyse

(siehe eCos Fallstudie)

- Klassische Umsetzung der Konfigurationsentscheidungen in den Komponenten mit Hilfe von #ifdef und Makros
 - Schutz vor ungewollten Ersetzungen nur durch strikte Namenskonvention
- **mangelnde Trennung der Belange**
 - viel Konfigurierungswissen ist im Quellcode verankert
 - quer schneidende Belange blähen die Funktionen auf
 - bedingte Übersetzung macht den Code schwer verständlich, zu warten und wiederzuverwenden

© 2006 Olaf Spinczyk

2

Lösungsansätze

(durch statische Konfigurationswerzeuge)

- Frame Prozessoren – z.B. **XVCL** [3]
 - Quellcode ist nur ein „Rahmen“, der je nach Konfiguration mit zusätzlichem Code angereichert werden kann
 - Ansatz ist unabhängig von der Programmiersprache
 - selbst Dokumentation lässt sich damit konfigurieren
- Variantenmanagement – z.B. **pure::variants** [1, 2]
 - zusätzliche (programmierbare) Ebene oberhalb der Komponenten
 - flexible Generierungs- und Transformationsmöglichkeiten
 - automatisierte Abbildung von Anforderungen (z.B. Merkmalselektion) auf Softwarevarianten



© 2006 Olaf Spinczyk

3

XVCL – ein *Frame* Prozessor

- XML-based Variant Configuration Language
- Was ist ein *frame*?

*„When one encounters a new situation (or makes a substantial change in one's view of a problem) one selects from memory a structure called a **frame**. This is a remembered framework to be adapted to fit reality by changing details as necessary.“*

M. Minsky

A Framework for Representing Knowledge

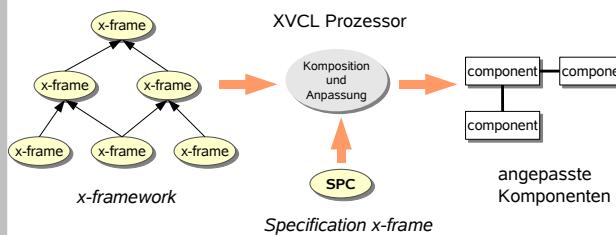
1975

- XVCL arbeitet mit *frames* in Form von Textdateien

© 2006 Olaf Spinczyk

4

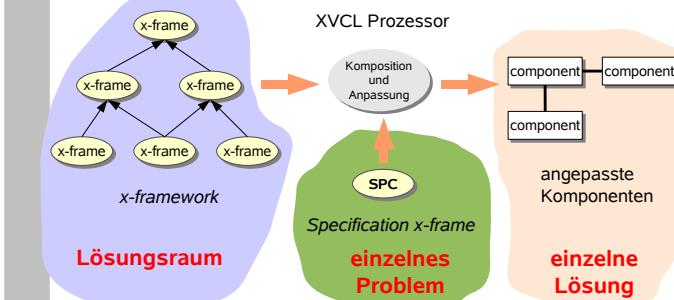
XVCL – Grundbegriffe



© 2006 Olaf Spinczyk

5

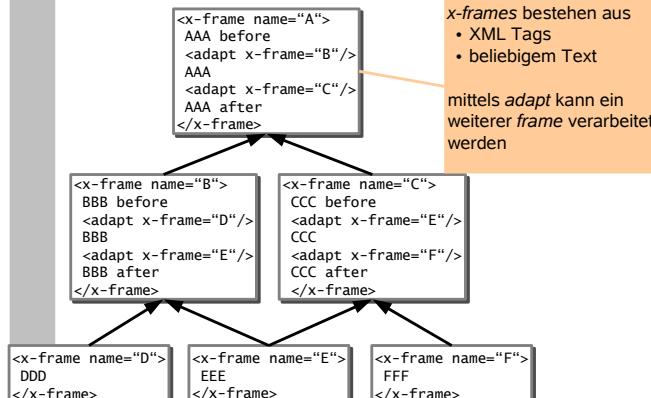
XVCL – Grundbegriffe



© 2006 Olaf Spinczyk

6

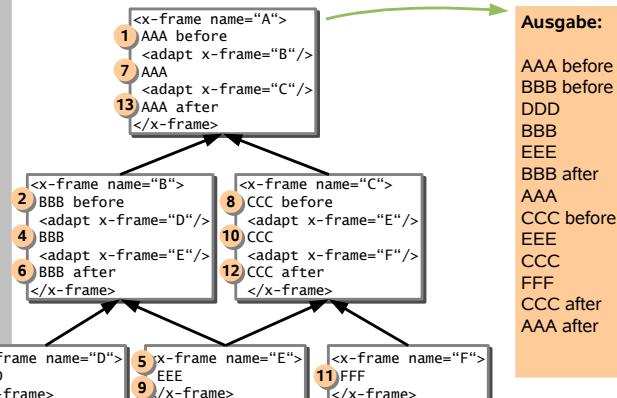
XVCL – Verarbeitung (1)



© 2006 Olaf Spinczyk

7

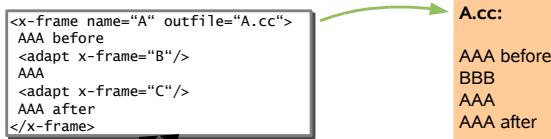
XVCL – Verarbeitung (2)



© 2006 Olaf Spinczyk

8

XVCL – Dateiselektion



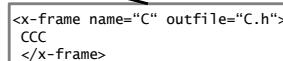
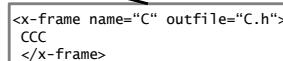
A.cc:
AAA before
BBB
AAA
AAA after

- normalerweise wirkt *adapt* wie `#include`
 - Komposition aus (angepassten) Codefragmenten
- in Verbindung mit *outfile* kann der SPC bestimmen, welche Dateien generiert werden



© 2006 Olaf Spinczyk

9



C.h:
CCC

9

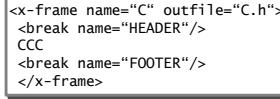
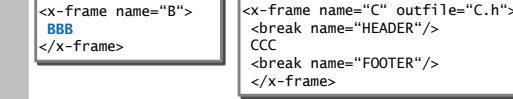
XVCL – Anpassung von Frames (1)

```

<x-frame name="A" outfile="A.cc">
  <adapt x-frame="C">
    <insert break="HEADER">
      --inserted--
    </insert>
    <insert break="FOOTER">
      <adapt x-frame="B"/>
    </insert>
  </adapt>
</x-frame>

```

C.h:
--inserted--
CCC
BBB



- durch *break* und *insert* (-before, -after) kann an vordefinierten Stellen durch übergeordnete *Frames* Text eingefügt werden

© 2006 Olaf Spinczyk

10

XVCL – Anpassung von Frames (2)

```

<x-frame name="A" outfile="A.cc">
  <set-multi var="NAME" value="a,b"/>
  <set var="VAL_a" value="4711"/>
  <set var="VAL_b" value="815"/>
  <while using-items-in="NAME">
    <set var="VAL" value="?@VAL_@NAME?"/>
    <adapt x-frame="C"/>
  </while>
</x-frame>

```

C.h:
a=4711
b=815

- Variablen, Schleifen und Bedingungen erlauben flexible Textgenerierung



© 2006 Olaf Spinczyk

11

XVCL – Beispiel I/O-Library (1)

```

<x-frame name="OStream">
  #ifndef __OStream_h__
  #define __OStream_h__

  #include "OStreamDock.h"
  <break name="STREAM_INCLUDE"/>

  class OStream : public OStreamDock {
  <break name="STREAM_ATTRIBUTE"/>
  public:
    OStream& operator &lt;&lt; (char) {...}
    OStream& operator &lt;&lt; (OStream& os) {...}
    <break name="STREAM_OP"/>
  };

  OStream& endl (OStream& os) {...}
  <break name="STREAM_MANIP"/>

  #endif // __OStream_h__
</x-frame>

```

- die OStream Klasse enthält lediglich das Grundgerüst und Markierungen an potentiellen Variationspunkten

© 2006 Olaf Spinczyk

12

XVCL – Beispiel I/O-Library (2)

```
<x-frame name="IntegerOStream">
<select option="SECTION">
<option value="INCLUDE">
#include "IntegerOutput.h"
</option>
<option value="ATTRIBUTE">
IntegerOutput iout;
</option>
<option value="OP">
OStream & operator &lt;&lt; (MaxInt i)
{ ... }
</option>
</select>
</x-frame>
```

- getrennte x-frames beschreiben die Erweiterungen für die verschiedenen unterstützten Datentypen (*slices*)



© 2006 Olaf Spinczyk

13

XVCL – Beispiel I/O-Library (2)

```
<x-frame name="IntegerOStream">
<select option="SECTION">
<option value="INCLUDE">
#include "IntegerOutput.h"
</option>
<option value="OP">
OStream & operator &lt;&lt; (void *p)
{ ... }
</option>
</select>
</x-frame>
```

...

- getrennte x-frames beschreiben die Erweiterungen für die verschiedenen unterstützten Datentypen (*slices*)



© 2006 Olaf Spinczyk

14

XVCL – Beispiel I/O-Library (3)

```
<x-frame name="ConfigOStream" outfile="OStream.h">
<adapt x-frame="OStream">
<insert break="STREAM_INCLUDE">
<set var="SECTION" value="INCLUDE"/>
<while using-items-in="TYPES">
<adapt x-frame="?@TYPES?"/>
</while>
</insert>
<insert break="STREAM_ATTRIBUTE">
<set var="SECTION" value="ATTRIBUTE"/>
<while using-items-in="TYPES">
<adapt x-frame="?@TYPES?"/>
</while>
</insert>
... weitere inserts für Operationen und Manipulatoren
</adapt>
</x-frame>
```

- ein weiterer x-frame fügt die OStream Klasse entsprechend der konfigurierten Typen in TYPES zusammen



© 2006 Olaf Spinczyk

15

XVCL – Beispiel I/O-Library (4)

```
<x-frame name="Test1">
<set-multi var="TYPES" value="IntegerOStream,PointerOStream"/>
<adapt x-frame="ConfigOStream"/>
</x-frame>
```

- der SPC beschreibt eine konkrete Konfiguration in kompakter Weise



© 2006 Olaf Spinczyk

16

XVCL - Zusammenfassung

■ Pro

- Konfigurationswissen und Programmcode kann getrennt werden
- selektives Generieren von Dateien möglich
- Quellcodeorganisation kann unabhängig von Modularisierungstechniken der Programmiersprache (z.B. Klassen) erfolgen
- programmiersprachenunabhängiger Ansatz

■ Contra

- XML Syntax und *Escape*-Zeichen im Quelltext, z.B. &
- fehlende Verbindung zwischen XVCL Quelltext und generiertem Quelltext problematisch bei Fehlermeldungen des Übersetzers
- **keine explizite Repräsentation des Problemraums**



© 2006 Olaf Spinczyk

17

Rollenverteilung bei der PL-Entwicklung

Problemraum

Domänenexperte

Lösungsraum

PL Architekt / PL Entwickler

Applikationsanalyst

Einzelnes Problem

Applikationsentwickler

Einzelne Lösung



© 2006 Olaf Spinczyk

18

Technische Herausforderung

Problemraum

Eigenschaften und Abhängigkeiten

Lösungsraum

Auswahl der Software-Elemente

Beschreibung
Realisierung

gewünschte Eigenschaften

Ressourceneffizienz

Einzelnes Problem

Einzelne Lösung



© 2006 Olaf Spinczyk

19

Variantenmanagement mit p::v

Problemraum

Featuremodell

Lösungsraum

Familienmodell

pure::variants

Variantenmodell

Einzelnes Problem

Variantenrealisierung

Einzelne Lösung



© 2006 Olaf Spinczyk

20

p::v Modellgrundstruktur

- Alle Modelle besitzen gleiche Grundstruktur
 - Element ist Grundbaustein
 - Elemente können gerichtete Beziehungen zu anderen Elementen haben ([Relationen](#))
 - Elemente können [Attribute](#) besitzen
 - An vielen Modellbestandteilen können [Restriktionen](#) die Gültigkeit einschränken
- Speicherung im XML-Format



© 2006 Olaf Spinczyk

21

p::v Element

- Jedes Element gehört zu einer Klasse:
ps:feature, ps:component, ps:part, ps:source
- Jedes Element hat einen Typ, z.B. ps:file
- Typsystem ist frei konfigurierbar, p::v benutzt fast immer nur die Elementklassen
 - Ausnahme: Standardtransformation
- Standardinformationen
 - ID, Unique Name, Visible Name, Beschreibung
- Optional: Relationen, Restriktionen



© 2006 Olaf Spinczyk

22

p::v Elementrelationen

- Elementrelationen sind „1:n“ Verbindungen
- Relationen werden immer im Ausgangselement gespeichert
- Relationen besitzen einen Typ
- p::v definiert einige Relationen wie z.B. „ps:requires“ (und deren Semantik)
- Benutzer kann eigene Relationstypen einführen
- Jede Relation kann eine Beschreibung besitzen
- Eine Relation kann Restriktionen besitzen
 - Relation ist nur gültig, wenn Restriktion „wahr“ liefert



© 2006 Olaf Spinczyk

23

p::v Elementattribute (1)

- Jedem Element kann eine beliebige Anzahl von Attributen zugeordnet sein.
- Ein Attribut ...
 - ist getypt (ps:string, ps:boolean, ...)
 - hat einen Namen
 - kann Restriktionen besitzen
- Der Attributwert ...
 - kann im Modell festgelegt werden (*fixed*)
 - kann in der Variantenbeschreibung gesetzt werden (*not fixed*)
- Beispiel: Merkmal Motor
 - (Attribut, Wert)₁ = („Leistung [kW]“, 85)
 - (Attribut, Wert)₂ = („Zylinder“, 4)



© 2006 Olaf Spinczyk

24

p::v Elementattribute (2)

- Attributwerte
 - Konstante
 - Berechnung (calculation)
- Jede Wertdefinition kann eine Restriktion besitzen
- Reihenfolge der Werte ergibt Berechnungsfolge
- Erste gültige Wertdefinition bestimmt Attributwert
- Ein Attribut eines ausgewählten Elements muss einen gültigen Wert haben
- „Not fixed“ Attribute können eine Defaultwertdefinition besitzen



© 2006 Olaf Spinczyk

25

p::v Restriktionen

- Eine Restriktion schränkt die Gültigkeit/Verwendbarkeit des zugeordneten Bestandteils ein
- p::v verwendet eine an OCL angelehnte Notation, die von pvProlog ausgewertet wird
- Die genaue Semantik wird durch die Art des Bestandteils bestimmt
- Beispiel:

```
hasFeature('A') or not(hasFeature('B'))
```



© 2006 Olaf Spinczyk

26

p::v Featuremodell (1)

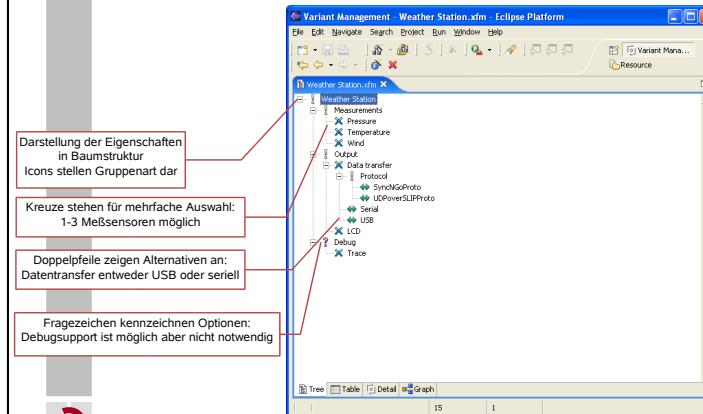
- Ein Featuremodell enthält nur Elemente der Klasse ps:feature
- Es gibt 4 Arten von Featuregruppen
 - notwendig (*ps:mandatory*) [n]
 - optional (*ps(optional)*) [0-n]
 - alternativ (*ps:alternative*) [1]
 - oder (*ps:or*) [1-n]*
- Jedes Feature kann von jeder Gruppenart maximal eine Gruppe besitzen



© 2006 Olaf Spinczyk

27

p::v Featuremodell (2)



© 2006 Olaf Spinczyk

28

p::v Featuremodell (3)

- Beziehungen zwischen Features (und auch anderen Modellelementen)
 - Gegenseitiger Ausschluß (*ps:conflicts*, *ps:discourages*)
 - Implikation (*ps:requires*, *ps:required_for*, *ps:recommends*, *ps:recommended_for*)
 - ... (erweiterbar)
- Semantik von 1:n Beziehung unterschiedlich
 - *ps:conflicts* und Co. : AND (Fehler, falls alle n selektiert sind)
 - *ps:requires* und Co. : OR (Fehler, falls keines der n selektiert ist)



© 2006 Olaf Spinczyk

29

p::v Familienmodell (1)

- Darstellung eines Lösungsraums
- Hierarchische Gruppierung von Elementklassen
 - Komponenten (*component*) enthalten Teile (*part*)
 - Teile enthalten weitere Teile und/oder Quellen (*source*)
- Teile und Quellen sind getypt
 - Typen werden in der Transformation ausgewertet
 - Typ gibt notwendige und optionale Attribute vor
- Restriktion ist Vorbedingung für Auswahl



© 2006 Olaf Spinczyk

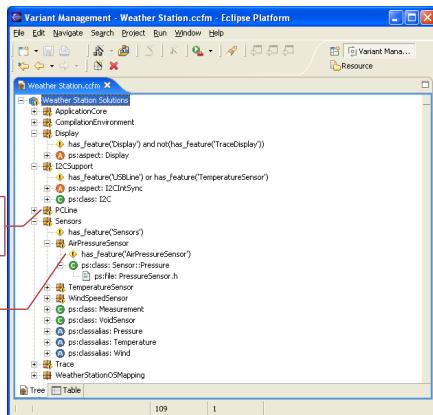
30

p::v Familienmodell (2)

Die Familienmodelle erfassen den Aufbau der Lösung von der abstrakten variablen Architektur bis hin zu den notwendigen Dateien. Neben der hier gezeigten Möglichkeit, die Informationen direkt in pure::variants zu bearbeiten, kann auch eine Koppelung mit anderen Modellierungswerzeugen erfolgen.

Die Icons stellen den Typ des Modellelements dar. Hier eine Komponente. Diese Typen sind frei definierbar (entsprechend der Architektur).

Diese Regeln steuern die Auswahl von Lösungselementen. Hier soll die Komponente „AirPressureSensors“ nur dann zur Lösung gehören, wenn das entsprechende Feature gleichen Namens gewählt wird.



© 2006 Olaf Spinczyk

31

p::v Konfigurationsraum (1)

(*configuration space*)

Zusammenstellung von Modellen für die gemeinsame Konfiguration

- Feature- und Familienmodelle können in mehreren Konfigurationsräumen verwendet werden
- speichert die Parameter für die (optionale) Transformation
- enthält beliebig viele Variantenbeschreibungen



© 2006 Olaf Spinczyk

32

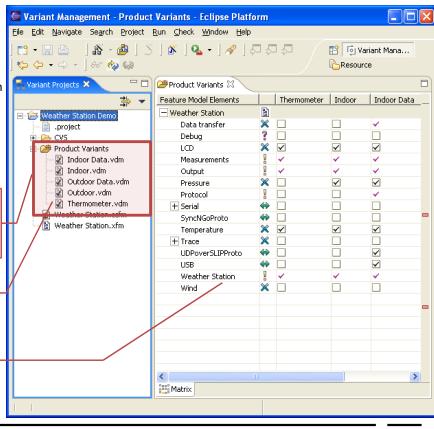
p::v Konfigurationsraum (2)

In pure::variants werden Produktvariabilitäten (Featuremodelle) und Lösungsarchitekturen (Familienmodelle) flexibel zu Produktlinien kombiniert. Eine Produktlinie kann aus beliebig vielen Modellen bestehen. Der „Configuration Space“ verwaltet diese Informationen und fasst die Varianten einer Produktlinie zusammen.

Der Configuration Space „Product Variants“ fasst die Variantendefinitionen für die Produkte zusammen.

Jedes Modell repräsentiert eine Variante

Diese Variantenmatrix ermöglicht einen Überblick über Unterschiede und Gemeinsamkeiten der Produktvarianten



33

© 2006 Olaf Spinczyk

p::v Variantenbeschreibung (1)

(variant description model)

definiert eine Variante aus den Möglichkeiten eines Konfigurationsraums

- enthält

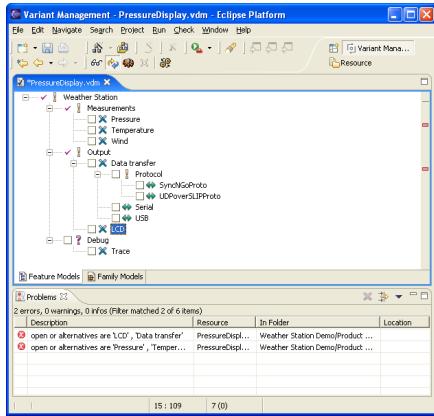
- alle gewählten Features
- alle durch Anwender spezifizierten Attributewerte

© 2006 Olaf Spinczyk

34

p::v Variantenbeschreibung (2)

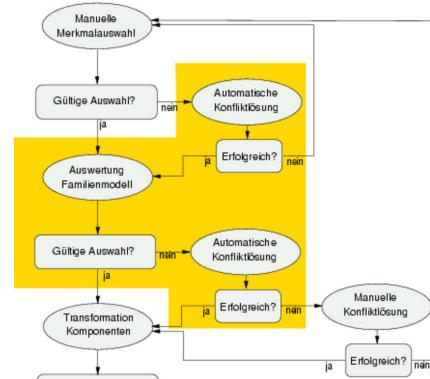
Die Variantenerstellung erfolgt durch Auswahl der gewünschten Features im Variantenmodell. pure::variants prüft die Auswahl interaktiv und löst oder meldet auftretende Probleme.



35

© 2006 Olaf Spinczyk

p::v Modellauswertung



© 2006 Olaf Spinczyk

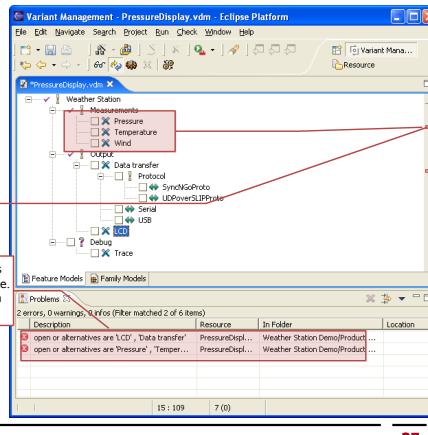
36

p::v Variantenerstellung

Die Variantenerstellung erfolgt durch Auswahl der gewünschten Features im Variantenmodell. pure::variants prüft die Auswahl interaktiv und löst oder meldet auftretende Probleme.

Durch einen Doppelklick wird zur Problemstelle navigiert.

Die Auswertung eines neuen Variantenmodells ergibt hier 2 vom Anwender zu lösende Probleme. In den beiden Mehrfachauswahlen wurde noch nichts ausgewählt.

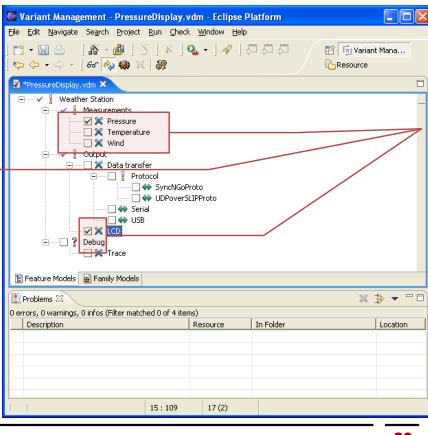


37

p::v Variantenerstellung

Die Variantenerstellung erfolgt durch Auswahl der gewünschten Features im Variantenmodell. pure::variants prüft die Auswahl interaktiv und löst oder meldet auftretende Probleme.

Die Auswahl von mindestens einem Feature (Pressure, LCD) löste das Problem.

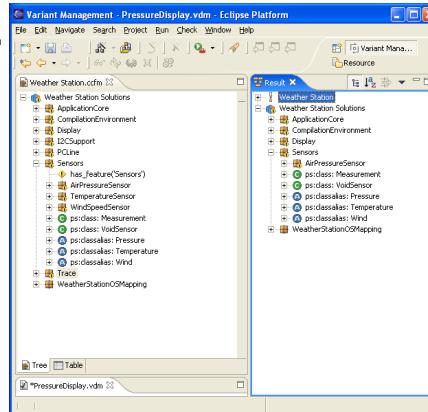


38

p::v Resultatsansicht

Die Resultatsicht (Result view) stellt die errechnete Lösung in (konfigurierbarer) Form als Baum oder Tabelle dar.

Export über das Kontextmenü möglich.



39

p::v Variantentransformation

Neben den Exportmöglichkeiten in verschiedene Formate wie HTML, CSV/Excel oder XML bietet pure::variants einen Mechanismus, mit dem aus Variantenmodellen die konkrete Lösung erzeugt wird.

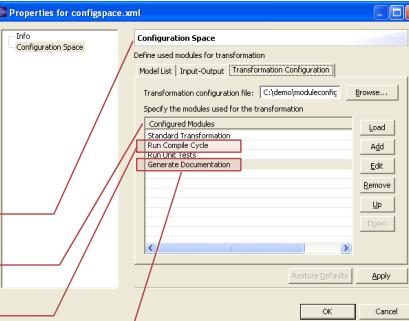
Dieser Transformationsmechanismus kann durch den Anwender beliebig erweitert und angepasst werden, sollten die Möglichkeiten der mitgelieferten Module nicht ausreichen.

Eine Transformationskonfiguration wird einem Configuration Space zugeordnet.

Die Module werden in der angegebenen Reihenfolge ausgeführt.

Die Einbindung externer Werkzeuge ist einfach und schnell möglich.

Mitgelieferte Module erlauben zum Beispiel die Erzeugung variantenspezifischer Dokumentation direkt aus Microsoft Word Dokumenten.



40

p::v Standardtransformationen

... erlauben im Wesentlichen:

- das Kopieren von Dateien in den Lösungsbaum
- das Zusammenstellen von Dateien aus Fragmenten
- das Erstellen von Links (nicht unter Windows)
- das Generieren von „Flag-Files“

```
#ifndef __flag_XXX
#define __flag_XXX
#define XXX 1
#endif // __flag_XXX
```

- das Generieren von „Class Aliases“

```
#ifndef __alias_YYY
#define __alias_YYY
#include "ZZZ.h"
typedef ZZZ YYY;
#endif // __alias_YYY
```



pure::variants - Zusammenfassung

- p::v ist ein als Eclipse-Plugin realisiertes kommerzielles Werkzeug zum Variantenmanagement
- p::v unterstützt ...
 - die Erfassung von Domänenwissen
 - die Beschreibung des Lösungsraums (die „Plattform“)
 - die Beschreibung konkreter Applikationsanforderungen
 - die Kontrolle der Lösung durch ein Resultatsmodell
- zur Auswertung von Attributen, Relationen und Restriktionen steht dem Entwickler die volle Leistungsfähigkeit von Prolog zur Verfügung
- die Generierung der Lösung basiert auf einer einfachen „Standardtransformation“, die aber erweitert und um weitere Transformationen ergänzt werden kann

pure::variants erlaubt das automatische Generieren von Applikationen anhand von Merkmalselektionen.

Ausblick

- Untersuchung verschiedener Techniken zur Umsetzung von Variabilität in der Implementierung der Komponenten
 - programmiersprachenbasierte Lösungen
 - Aspekte
 - Objekte
 - Templates
 - Mixin Layers



Literatur

- [1] pure-systems GmbH. *Variantenmanagement mit pure-variants*. Technical White Paper, <http://www.pure-systems.com/>.
- [2] pure-systems GmbH. *pure::variants Eclipse Plugin. pure-variants*. Manual, <http://www.pure-systems.com/>.
- [3] National University of Singapore. *XML-based Variant Configuration Language (XVCL)*. 2004.

