The PROTEUS Configuration Language

Reidar Conradi, Bjørn Gulla, Eirik Tryggeseth, NTH, Trondheim
Ian Sommerville, Lancaster University, Lancaster
Bjorn Gronquist, CAP Gemini Innovation, Grenoble

Overview

1. PCL
   • Background
   • Entity declarations
   • System modelling levels
   • Classifications
   • Attributes
   • Relations: predefined and user-defined
   • Physical mapping
   • Derivation and installation
2. PCL support tools
The PROTEUS project

- ESPRIT-3 joint european project, #6086
- Objective: provide a methodology and a toolset to support systems evolution
- Consortium:
  - CAP Gemini Innovation, Grenoble, France
  - CAP debis GEI, Aachen, Germany
  - Lancaster University, Lancaster, UK
  - Matra Marconi Space, Toulouse, France
  - SINTEF-DELAB/NTH, Trondheim, Norway
  - CAP SESA Telecom, Rennes, France
  - HP, Grenoble, France
  - INTECS, Pisa, Italy
- Started July 1992
- Total effort: approx. 60 person-years

Background

- Background: MIL, major SCM systems (Adele, DSEE etc.), E-R modelling, O-O.
- Based on existing technology: Make tools, RCS/CVS version handling systems, BMS message services. Loosely integrated tools.

- Requirements for PCL
  - document a system model: PCL should provide facilities for modelling domain concepts, families of software systems, and individual system instances
  - control system building: PCL should provide facilities to allow automated system building and installation on hardware platforms
**Entity declarations**

<table>
<thead>
<tr>
<th>Name</th>
<th>Abstraction</th>
<th>Type</th>
<th>Parent</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proteus-str-viewer</td>
<td>family/system software</td>
<td>STRUCTURED-</td>
<td>refines</td>
<td></td>
</tr>
</tbody>
</table>

**attributes**
- LANGUAGE_VIEWER: \( boolean := \) true
- PLATFORM_SELECTION: \( (HP, \ Sun, \ Macintosh, \ PC) := \) HP
- OS_SELECTION: \( (Unix, \ MacFinder, \ DOS) := \) Unix

**interface**

**processes**
- SETUP_ROUTINE => LIBSS/SETUP_ROUTINE
  - if LANGUAGE_VIEWER = TRUE then
  - COMPILER
  - end if
- INTERFACE => SPECIAL_INTERFACE
- EDITOR
- DATABASE

**relationship**
- physical repository "proj/proteus/viewer"

---

**System modelling levels**

- **Generic level**
  - describe abstractions from a particular domain
  - define structure and attributes, but no actual realization components

- **Family level**
  - single description of all versions of an evolving system or a part thereof
  - may include references to physical implementation components and derivation tools

- **Instance level**
  - all software and hardware bindings instantiated
Entity classifications

- Several dimensions for classification, each strict hierarchy
- Used for physical mapping, derivation-rule association, and relation definitions

Example of 'type' dimension

Predefined
User's extensions
Hardware
Software
**Attributes**

Bound attributes define properties of possible instantiations

Unbound attributes define dimensions of variability of possible instantiations

- **Examples:**
  - `manufactured_by := "Sun Microsystems, Inc."`
  - `options := if fast then "-f" end if`
  - `disk-size: (Mb40, Mb80, Mb120) := Mb80`
  - `speed : (slow, medium, fast) := imported fast`

- **Domains:** boolean, integer, enumeration, string

- **Binding:**
  - define-time
  - build-time

---

**Relationships**

Built-in
- `part_of (composition)`
- `uses (dependency)`
- `instance_of`
- `refines (concept-to-realization, bind variability)`
- `platform, installation`

User-defined
- `defined in terms of classifications`

```plaintext
relation is-derived-from
  domain design-document
  range source-code
end

family BergenCC
...
relationships
relate is-generated-from (BergenCCdesign)
end
```
Facilities to express variability are needed because of:

- **Evolution.** Development and maintenance activities: corrections, enhancements, restructuring, porting; planned or un-planned
- **Variants.** Multi-platform applications, different operational requirements, customer tailoring

**Support in PCL:**
- unbound (build-time) attributes
- conditional expressions
- slot-binding mechanism for relationship versioning
- refinement relation
- flexible physical mapping
- intensional version selection

**Types of variability:**
- structural variability
- realization variability
- component version selection
- conditional ‘compilation’
- derivation tool selection and composition
- tool parameters
- platform, run-time organization, distribution

**Physical mapping**

- M:N mapping
- powerful implicit mechanisms

**The physical mapping process:**
1. physical component name
2. repository path
3. version selection
4. workspace path
Derivation rules

<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
<th>Classification</th>
<th>Parent</th>
<th>Relation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lex</td>
<td>derivation-rule</td>
<td>tool</td>
<td>lexical-analyser</td>
<td>refines</td>
</tr>
</tbody>
</table>

attributes
-- The tool switch inherited from 'lexical-analyser'
-- Now: Attributes to model the appearance in the build script
Lex-Flags: string := constant
  if fast = TRUE then "-f" endif
  if quiet = TRUE then "-n" endif
  if verbose = TRUE then "-v" endif
Lex-Out: string := constant
  if to-stdout = TRUE then "-t" ++ " > " ++ c-source.name endif
Lex-Cmd: string := constant
  if to-stdout = TRUE then "-t" ++ " > " ++ c-source.name endif

in
-- Inherited from lexical-analyser
out c-source
platform sun4
build-script Lex-Cmd ++ Lex-Flags ++ lex-source.name ++ Lex-Out
depend-script

Figure 1: Basic Model

PCL-Decs PCL-Derivers

The component’s default physical mapping

classification lex-source is a source-code
physical
  name=".l"
end

tool supplier=>"AT&T"

component pascal-lexer
type : lex-source
end

derivation-rule lex is
attributes
  tool supplier=>"AT&T"
in
  lex-source
end

Togethers determines the derivation rule to be used.

Current PCL description

Figure 1: Model for file selection and derivation rule binding.
PCL tools

- **PCL-edit**: graphical browsing and editing of PCL descriptions, view and searching facilities, invocation of other functions/tools
- **PCL-analyze**: parsing of textual representations, AST and name space management
- **PCL-bind**: determining a unique instance, removing PCL-internal variability: bind build-time attributes, evaluate conditional expressions, bind slots, identifies physical components
- **PCL-select**: version selection of physical components
- **PCL-makegen**: associate derivation rule to product parts, generate makefiles
- **PCL-makeinstall**: generate install files
- **PCL-make**: establish workspace, generate derived elements ('binaries'), exploiting previously derived elements where possible
- **PCL-install**: download application parts onto hardware/run-time platform (execution possibly inter-leaved with PCL-make)
- **Repository Interface**: uniform interface to underlying versions management systems, includes arbitrary attribute annotations

Architecture

![Architecture Diagram]

Design-tool

Design description

Translator

PCL - description

PCL-edit

PCL-edit-UI

Repository-UI

Component Selection

P-makefile

P-install-file

PCL-build

PCL-build-UI

Component Check-in/out

Repository Interface
Figure 2: The evaluation process
Conclusions

• Major contributions:
  • integrated hardware and software configuration
  • heterogenous, distributed applications
  • modelling of stable and variable parts
  • evolution and variant support
  • integrated product and derivation process modelling
  • interactive, graphical tool support

• Current state:
  • Requirement analysis and specifications completed
  • PCL toolset due for preliminary review in June
  • Subsequent industrial evaluation of language and tools