From design to manufacturing: a product definition based on a pythonOCC™/STEP framework.

11th NASA-ESA Workshop on Product Data Exchange
Kent Space Center, 29 April - 1 May 2009

Thomas Paviot*, Jelle Feringa*, Stephen Waterbury**
*pythonOCC project: tpaviot@gmail.com; jelleferinga@gmail.com
**NASA/GSFC: stephen.c.waterbury@nasa.gov
Table of content

- Context / Need
- The design-logistics/manufacturing interoperability issue
- A suitable framework architecture
- Limitations from commercial CAD application / need for an open source industrial quality 3D CAD framework
- The pythonOCC project
- pythonOCC demos
- Questions
Context/Need: complex products, extended enterprise

- A bridge between design and manufacturing in order to:
  - Reduce time/errors for BOM transfer (EBOM ↔ MBOM),
  - Automate routings creation and optimization, flatten MBOM levels,
  - Propagate changes from design to manufacturing (or the opposite),
  - Optimize/design logistics support/maintenance.
Integrated design/production: an interoperability issue

How can we make these different tools exchange information knowing that:

- They have different functions and semantics,
- Used by different people, with different skills and expectations,
- Sold by vendors that compete on (almost) all these markets and do not want to interoperate with product of the competition!

ERP (Enterprise Resource Planning)

Functions: sourcing/logistics, supply chain management, production planning, MRP, order forecasting etc.

Data: parts, products, routings, suppliers, work centers, BOM, ECO/ECR etc.

3D CAD

Functions: geometric 3D definition, 2D drafts, NC manufacturing etc.

PDM (Product Data Management)

Functions: design engineering, sourcing/logistics, digital supply chain management, production planning, MRP, order forecasting etc.

Data: parts, products, CAD documents, team, roles, document maturity, etc.

Real product area

Virtual product area
**STEP and semantic web - SOA architecture**

- Knowledge database RDF/OWL
- SOA ERP (Sage, SAP R3, OpenERP etc.)
- SOA PDM (PTC Windchill, DS MatrixOne etc.)
- PLMServices 2.0-based SOA public ring
- STEP AP239 (PLCS) data model
- CAD modeling tool
- Detailed in the next part of the presentation

Reasoning results/SPARQL queries

Knowledge populating (manufacturing info)

Thomas Paviot Ph.D. research (PLM09 conf. - July 2009, Bath University - G.B.)

April 2009, 29th

11th NASA/ESA Workshop on Product Data Exchange - pythonOCC/STEP-PLCS framework
Issues related to commercial CAD programs

- Need for an access to low level topology (for instance, connect a face with a knowledge information such as dimensions, the machine required for manufacturing, reference face etc.): commercial CAD app scripting features (VBA for instance) do not allow to access this low level topology.

- The access to the complete API of commercial apps is highly expensive: difficult to justify this cost for an experimental development,

- Commercial CAD products API require C++ programming (not the best choice actually for application prototyping),

- The subsequent code licensing maybe difficult to understand or limitative (terms of the redistribution to other research teams?)

- Implementation of open standards (IGES, STEP) may be uncomplete, causing data exchange issues.

Need of a low-cost, industrial quality, easy-to-use, maintain and deploy 3D CAD kernel.
The pythonOCC project

Why OpenCASCADE™?
- The only industrial quality 3D CAD kernel,
- OpenSource,
- But: C++, difficult to start with.

Why Python™?
- Industrial quality scripting language,
- OpenSource, portable
- Widely used in scientific computing (Salomé, Code Aster etc.), many useful open source libraries: VTK, pyTables, numpy, SciPy etc.
- But: no available CAD library

Why pythonOCC?
- to quickly develop/prototype industrial quality CAD/PLM portables applications

An open source/open standards framework dedicated to agile CAD development.
Project history / status

- Started in April 2008,
- First usable release in February 2009,
- Latest release: pythonOCC 0.2 (2009, April 10th),
- Available for both Windows XP/Vista, MacOSX 10.5, GNU Linux (tested with Debian, Fedora, OpenSuse and Ubuntu distributions),
- Distributed under the free and open source GNU General Public License v3,
- Collaborative development platform (Subversion repository, mailing-list, bugtracker etc.) hosted by gna: http://gna.org/projects/pythonocc
- Website, wiki, API reference online documentation, downloads: http://www.pythonocc.org
- More than 9000 classes covered by the wrapper (about 90% of the OCC library except WOK),
- A binding generation system made of 3 python scripts = no more than 3k lines of code → easy maintenance,
- Many tutorials/samples available,
- A growing users community.
A two level architecture for agile CAD development

- pythonOCC aims to extend the 3D modeling features of the OCC kernel with the newest developments in the field of Knowledge Based Engineering, collaborative work, product representation etc.
- pythonOCC uses other well-know scientific libraries

L0: OpenCascade C++ library
L1: Python bindings, Traits, multi processing, twisted, ...
L2: High level CAD/PLM pythonic API

pythonOCC
Level 1: a simple but fully automated binding generation system

Process input:
OpenCascade 6.3.0 headers (14197 files)

Process output:
Python bindings (*.py and *.pyd/*.so libraries)

Build process (g++/msvc)

XML files

pygccxml processing

Set of *.i files
- Adaptor2d.i  13 Ko
- Adaptor2d_dependencies.i  6 Ko
- Adaptor2d_headers.i  2 Ko
- Adaptor3d.i  51 Ko
- Adaptor3d_dependencies.i  8 Ko
- Adaptor3d_headers.i  4 Ko
- AdvApp2Var.i  36 Ko
- AdvApp2Var_dependencies.i  6 Ko
- AdvApp2Var_headers.i  4 Ko

Build process (g++/msvc)

Set of *.cpp and *.py files
- BinMDData2d_headers.i  5 Ko
- BinMDData3d_wrap.cpp  1200 Ko
- BinMDData3d_Wrapper.i  7 Ko
- BinMDF.i  25 Ko
- BinMDF_dependencies.i  6 Ko

Py++ processing

SWIG processing
Level 2: a set of high-level classes/methods and programming samples

Mid-term objectives:
- A KBE systems based on ontologies and SPARQL,
- A complete collaborative design environment: asynchronous collaboration via PDM storage (with a **small** granularity database), synchronous collaboration (3D realtime broadcasting, chat system).
pythonOCC demos

- These demos are intended to show that:
  - The Level1 API (L1 demos) offer all OpenCascade features,
  - The Level2 API (L2) extend OCC features by using well known python libraries,
  - Python scripting is a practical and quick way to access OCC features (agile CAD development),
Demo 1: an easy to use/flexible scripting engine (L1) for 3D modeling

```python
from OCC.BRepPrimAPI import *
box = BRepPrimAPI_MakeBox(10,20,30).Shape()
display.DisplayShape(box)
```
Demo2 : standard file handle / visualization (L1)

- The pythonOCC CADViewer for IGES/BRep/STL/STEP files was achieved with only 80 lines of python code

STEP file visualization (aircraft)  IGES file visualization (fan)
Demo 3: boolean operations (L1)

- The ‘Emmenthaler’ script performs boolean operations: it removes 50 random cylinders from an initial 200*260*260 box.
Demo 4 : gaussian curvature viewer (L2)

- This sample exploits high level Topology API to compute/display the gaussian curvature of an IGES geometry (about only 250 lines of code).
Demo 5 : multiprocessing (L2)

- Rarely geometric processes run in parallel, but pythonOCC with the multiprocessing module facilitate this task,
- Useful with design-of-experiments and generative design,
- Speeds up generation of complex geometry dramatically,
- Example: exploit all available processor cores (8) to slice complex geometry,
- Near linear speed up for the following example: finding multiple intersections between a geometry and parallel planes (get for instance the tool path for rapid prototyping). ‘Slicing’ example.
Demo 5: multiprocessing results

Multiprocess performance results
(MacOSX 10.5/8 processors machines/ pythonOCC svn rev. 302/ STEP geometry)

10,000 slices performance test

Time(s)

Number of processors
Demo 6: network collaboration - geometry sharing

Geometry exchange (sphere) over a network via XML/RPC. 2 scripts:
- client.py
- server.py

Diagram:
- Geometry factory server
- XML/RPC server
- Geometry Display
- XML/RPC client
- Machine 1 (geometry creation - Modeling)
- Machine 2 (geometry client - Display)
Demo 6: 1 shape server, 3 concurrent clients
Thank you for your attention!

Any question?
Please email us for further information.

tpaviot@gmail.com      jelleferinga@gmail.com