Formal Management of CAD/CAM Processes

Michael Kohlhase Johannes Lemburg Lutz Schröder
Ewaryst Schulz

DFKI Bremen, Germany

IFIP WG 1.3. Meeting, Udine, Sept. 2009

Sichere Kognitive Systeme

Introduction

- Analogies between engineering design process and software engineering processes exist theoretically.
- In practice, final ‘implementation’ step dominant
  - has well-developed tool support: CAD systems
- Idea: break this dominance by providing an integrated development methodology
  - formal, semi-formal, informal documents
  - tool support at all levels
  - invasion into CAD tools to provide user interfaces
- Context: FormalSafe project at DFKI
  - Comprehensive framework for document-oriented development process
- Today: experiments in formal verification of CAD objects (FM 09)

Why?

Why?

- Formal verification of physical properties
- Tracing of (formalized) requirements
- Improved control over the coherence of designs
- Semantically founded change management.

Systematic Engineering Design ('Konstruktionslehre')

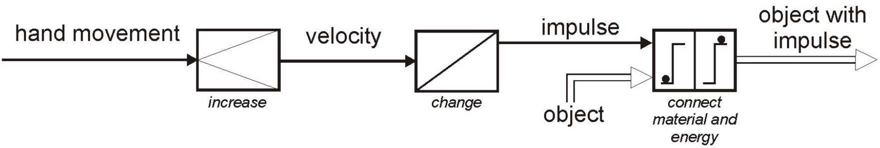
System of design stages:

- Requirements
- Function structure
- Principle solution
- Embodiment

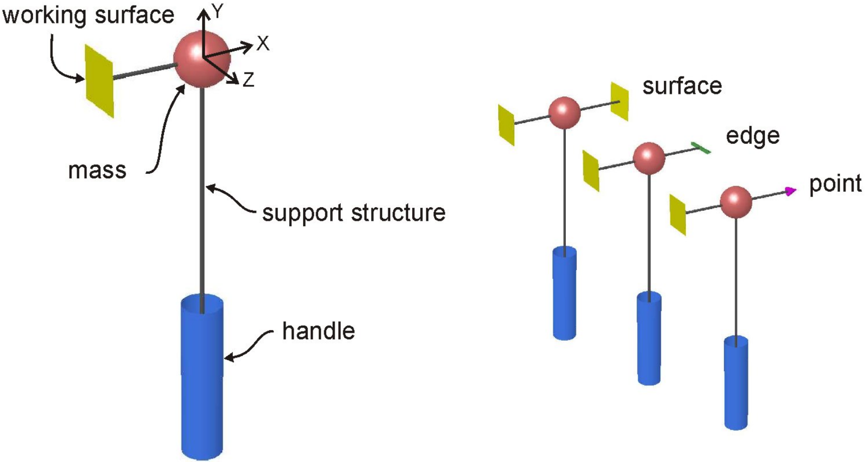
Requirements on a Hammer

*A **hammer** is an apparatus for the manual generation and transmission of a defined impulse to an object, e.g. for driving a nail into a wall.*

The Function Structure of a Hammer



The Principle Solution for a Hammer



The Embodiment of a Hammer

Invading SolidWorks

- Invasive approach:
  Direct access to data structures of the CAD system
- Plug-in programmed using the SolidWorks API
- Presently: export of CAD objects into HasCasl.

Modelling CAD Objects

- Regard CAD data structures as syntax, modelled as an algebraic datatype
- Provide a background modelling of abstract geometry, such as affine real geometry
- Define the semantics of CAD objects as point sets in affine space $\mathbb{R}^3$
- Verify geometric properties of objects

The Syntax of CAD Objects

spec SOLIDWORKS = AFFINEREALSPACE3DWITHSETS
then free types

$$
\begin{aligned}
&SWPlane ::= SWPlane\,(p : Point;\ normal : VectorStar;\\
&\qquad\qquad innerCS : Vector);\\
&SWArc ::= SWArc\,(Center : Point;\ Start : Point;\ End : Point);\\
&SWLine ::= SWLine\,(From : Point;\ To : Point);\\
&SWSpline ::= SWSpline\,(Points : List\ Point);\\
&SWSketchObject ::= type\ SWArc \mid type\ SWLine \mid type\ SWSpline;\\
&SWSketch ::= SWSketch\,(Objects : List\ SWSketchObject;\\
&\qquad\qquad Plane : SWPlane);\\
&SWExtrusion ::= SWExtrusion\,(Sketch : SWSketch;\ Depth : Real);\\
&\ldots\\
&SWFeature ::= type\ SWExtrusion \mid \ldots
\end{aligned}
$$

Modelling Affine Geometry

spec AFFINESPACE[VECTORSPACE[FIELD]] $=$

type $Point$

op $\_\_+\_\_ : Point \times Space \rightarrow Point$ %(point space map)%

vars $p, q : Point; v, w : Space$

$\bullet\ p + v = p + w \Rightarrow v = w$ %(plus injective)%

$\bullet\ \exists\, y : Space \bullet p + y = q$ %(plus surjective)%

$\bullet\ p + (v + w) = p + v + w;$ %(point vector plus associative)%

then %implies

$\forall\, p : Point; v, w : Space$

$\bullet\ p + v + w = p + w + v;$ %(point vector plus commutative)%

end

spec EXTAFFINESPACE [AFFINESPACE[VECTORSPACE[FIELD]]] $=$ **%def**

op $vec : Point \times Point \rightarrow Space$

$\forall\, p, q : Point \bullet p + vec\,(p, q) = q;$ %(vec def)%

Defining CAD Semantics (Extract)

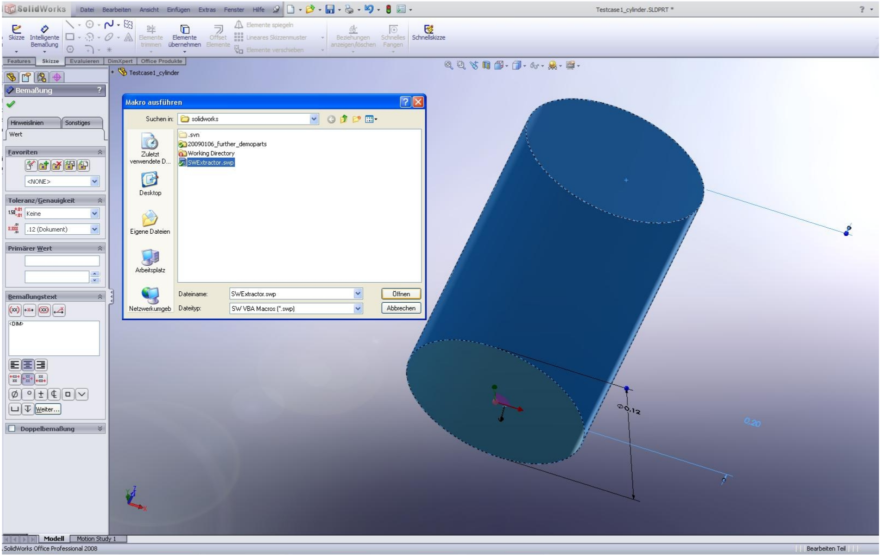
ops $VWithLength(v : Vector; s : Real) : Vector =$
$v \text{ when } v = 0 \text{ else } (s \,/\, (\| v \| \text{ as } NonZero)) * v;$
$ActExtrude(ax : Vector; ps : PointSet) : PointSet =$
$\lambda\, x : Point \bullet \exists\, l : Real; y : Point$
$\bullet\, l \text{ isIn closedinterval } (0, 1) \wedge y \text{ isIn } ps \wedge x = y + l * ax;$
$i : SWExtrusion \rightarrow PointSet;$
$i : SWPlane \rightarrow PointSet$
$i : SWSketch \rightarrow PointSet;$
$is : SWSketchObject \times SWPlane \rightarrow PointSet;$
$is : (List\ SWSketchObject) \times SWPlane \rightarrow PointSet;$

vars $o, x, y, z : Point; n : VectorStar; ics : Vector; l : Real;$
$sk : SWSketch; plane : SWPlane;$
$sko : SWSketchObject; skos : List\ SWSketchObject$

$\bullet\ is\ (sko :: skos, plane) = is\ (sko, plane) \text{ union } is\ (skos, plane);$
$\bullet\ i\ (SWExtrusion\ (sk, l))$
$= ActExtrude(VWithLength\ (normal\ (Plane\ sk), l), i\ sk);$

Exporting a Cylinder: Before

Sichere Kognitive Systeme



Exporting a Cylinder: After

Slightly abstracting, have something matching the following pattern:

spec SolidWorksCylByArcExtrusion =
SolidWorksPlane_is_AffinePlane
then op

$$SWCylinder(center, boundarypt : Point; axis : NZVector) : SWFeature =$$

$$\textit{let } plane = SWPlane\ (center, axis, V\ (0, 0, 0));$$
$$arc = SWArc\ (center, boundarypt, boundarypt);$$
$$height = \|\ axis\ \|$$
$$\textit{in } SWExtrusion\ (SWSketch\ ([\ arc\ ], plane), height);$$

Correctness of the Cylinder Design

$$
\begin{array}{l}
\textbf{spec}\ \textsc{Cylinder} = \textsc{AffineRealSpace3DWithSets} \\
\textbf{then op}\quad \mathit{Cylinder}(\mathit{base} : \mathit{Point}; r : \mathit{PReal}; \mathit{ax} : \mathit{NZVector}) : \mathit{PointSet} = \\
\qquad \lambda\ x : \mathit{Point} \bullet \mathit{let}\ v = \mathit{vec}\ (\mathit{base}, x)\ \mathit{in} \\
\qquad\qquad || \mathit{proj}\ (v, \mathit{ax})\ || \leq ||\ \mathit{ax}\ || \\
\qquad\qquad \wedge\ ||\ \mathit{orthcomp}\ (v, \mathit{ax})\ || \leq r \\
\qquad\qquad \wedge\ (v * \mathit{ax}) \geq 0;
\end{array}
$$

$$
\begin{array}{l}
\textbf{view}\ \textsc{SWCylByAE\_IsCylinder} : \textsc{Cylinder}\ \textbf{to} \\
\quad \{\textsc{SolidWorksCylByArcExtrusion} \\
\quad \textbf{then op} \\
\qquad \mathit{Cylinder}(\mathit{base} : \mathit{Point}; r : \mathit{PReal}; \mathit{axis} : \mathit{NZVector}) : \mathit{PointSet} = \\
\qquad \mathit{let}\ \mathit{boundary} = \lambda\ p : \mathit{Point} \bullet \mathit{let}\ v = \mathit{vec}\ (\mathit{base}, p) \\
\qquad\qquad\qquad \mathit{in}\ \mathit{orth}\ (v, \mathit{axis}) \wedge ||\ v\ || = r; \\
\qquad\quad \mathit{boundarypt} = \mathit{choose}\ \mathit{boundary} \\
\qquad \mathit{in}\ i\ (\mathit{SWCylinder}\ (\mathit{base}, \mathit{boundarypt}, \mathit{axis})); \\
\quad \}
\end{array}
$$

Conclusion

- Export to HasCasl turns CAD objects into fully formal documents
  - Amenable to formal verification
  - Semantic change management
  - Rapid prototyping
- Proof of concept: formal verification of a cylinder
  - Several hundred lines of Isabelle/HOL
- Open issues:
  - Scalability
    - Make use of modularity
    - Library of patterns
  - Methodology
- Next: verify designs against principle solutions, e.g. for the hammer.