Center for *e*-Design

Real Time Collaboration and Sharing

National Science Foundation Industry/University Cooperative Research Center for *e*-Design:
IT-Enabled Design and Realization of Engineered Products and Systems

A Programming Language Approach to Parametric CAD Data Exchange

Presented by: John Altidor

Jack Wileden, Jeffrey McPherson,

Ian Grosse, Sundar Krishnamurty,

Felicia Cordeiro, Audrey Lee-St. John

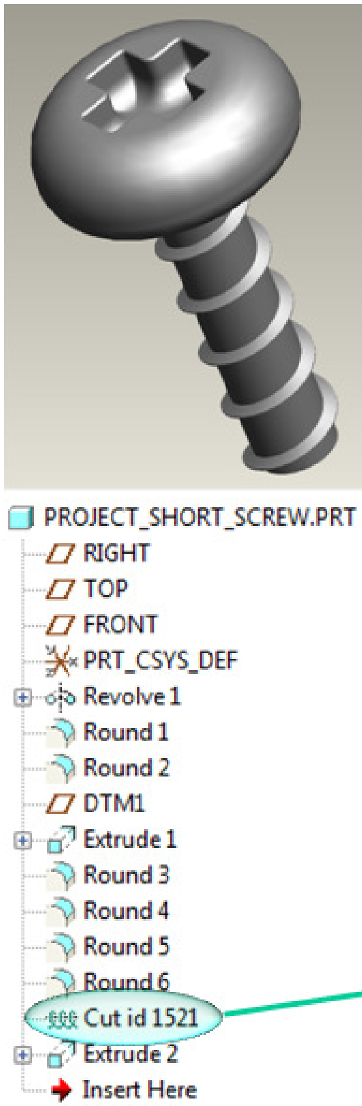
UMass Amherst

MOUNT HOLYOKE

Problems in CAD Exchange

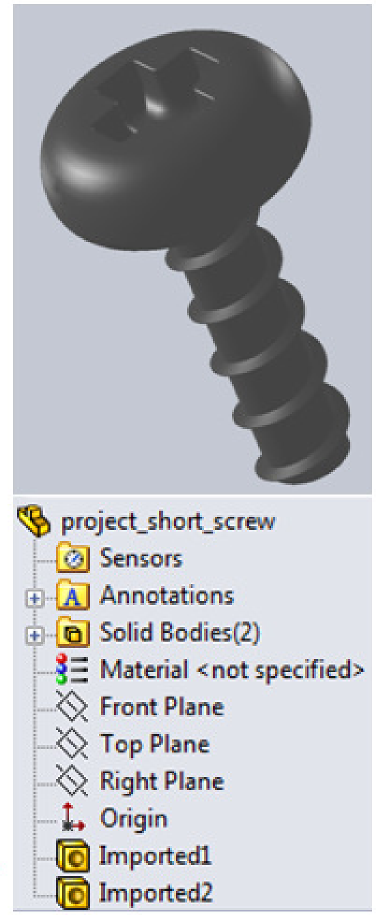
- Data exchange between different computer-aided design (CAD) systems is a major problem inhibiting integration.
- Existing standard CAD formats (e.g. IGES, STEP) are inadequate for:
  - Modification
  - Extension
  - Higher-level functionality
- Exchanging **features** generally requires **manually recreating** a CAD model.

Features Lost



IGES File Format

- Geometric information maintained
- Features cannot be selected
- Design intent gone
- In this example, pitch (number of threads) cannot be changed



Related Work for Preserving Features

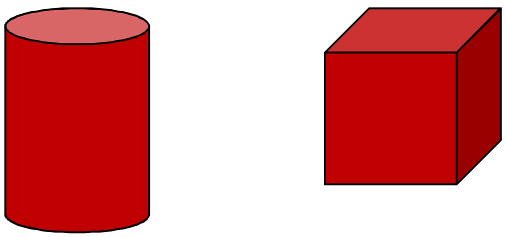
- **Commercial software for CAD translations.**
  - Proficiency
  - Translation Technologies
  - Elysium
- **Ontological approaches to CAD translations.**
  - Zhu et al. “Ontology-driven integration of CAD/CAE application: Strategies and Comparisons”.
  - Patil et al. “Ontology-based exchanged of product design semantics”.
- **See paper for comparison.**

Language Approach

- CAD systems are modeled as **programming languages**.
- CAD models correspond to **programs** in the languages modeling the CAD systems.
- Case studies applying our models, methods and tools to popular CAD systems (**Pro/Engineer** and **SolidWorks**) to assess and guide our research.

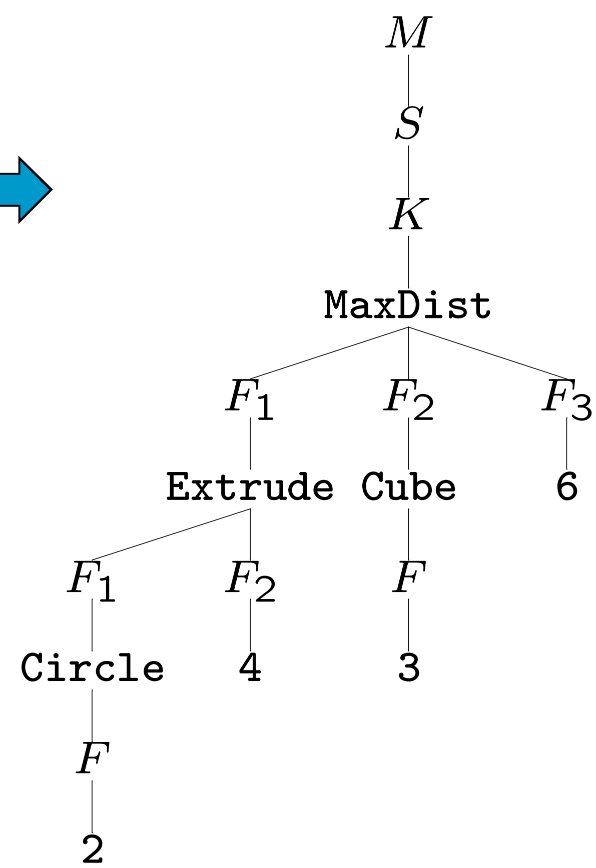
CAD Systems Background

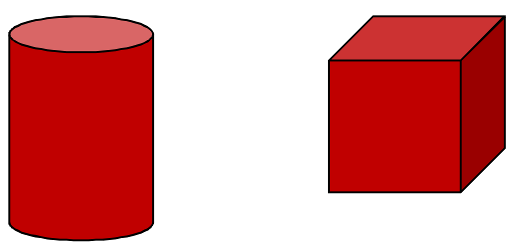
1. Create a 2-dimensional (2D) circle with radius of 2cm.
2. Extrude the circle with depth of 4cm.
3. Create a cube with side-length 3cm.
4. Add a constraint to make centers of the cylinder and cube be separated by exactly 6cm.



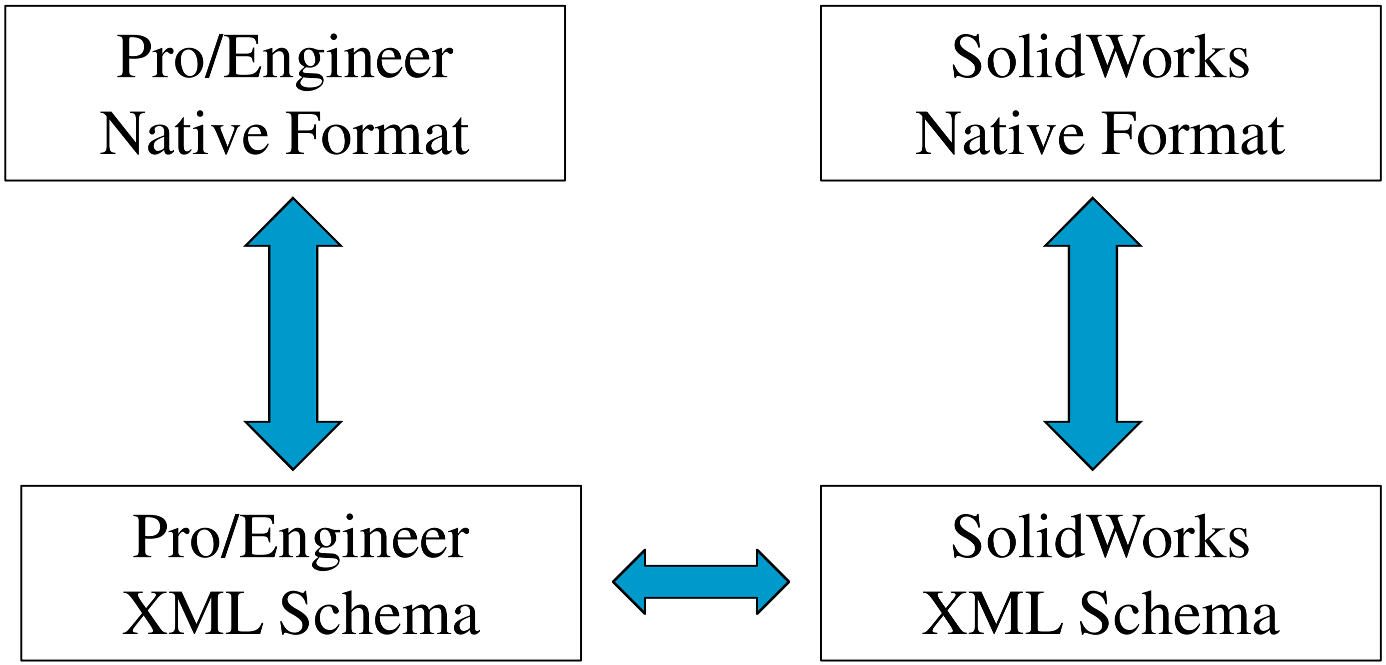
CAD System as Languages

Category			AST Node
Model	M	::=	S | `Sequence`(S, M)
Statement	S	::=	F | K | `Assign`(x, F)
Feature	F	::=	n | x | `Extrude`(F_1, F_2)
		|	`Cube`(F)
		|	`Circle`(F)
		|	`Triangle`(F_1, F_2, F_3) | ...
Constraint	K	::=	`Boundary`(F_1, F_2)
		|	`MaxDist`(F_1, F_2, F_3)

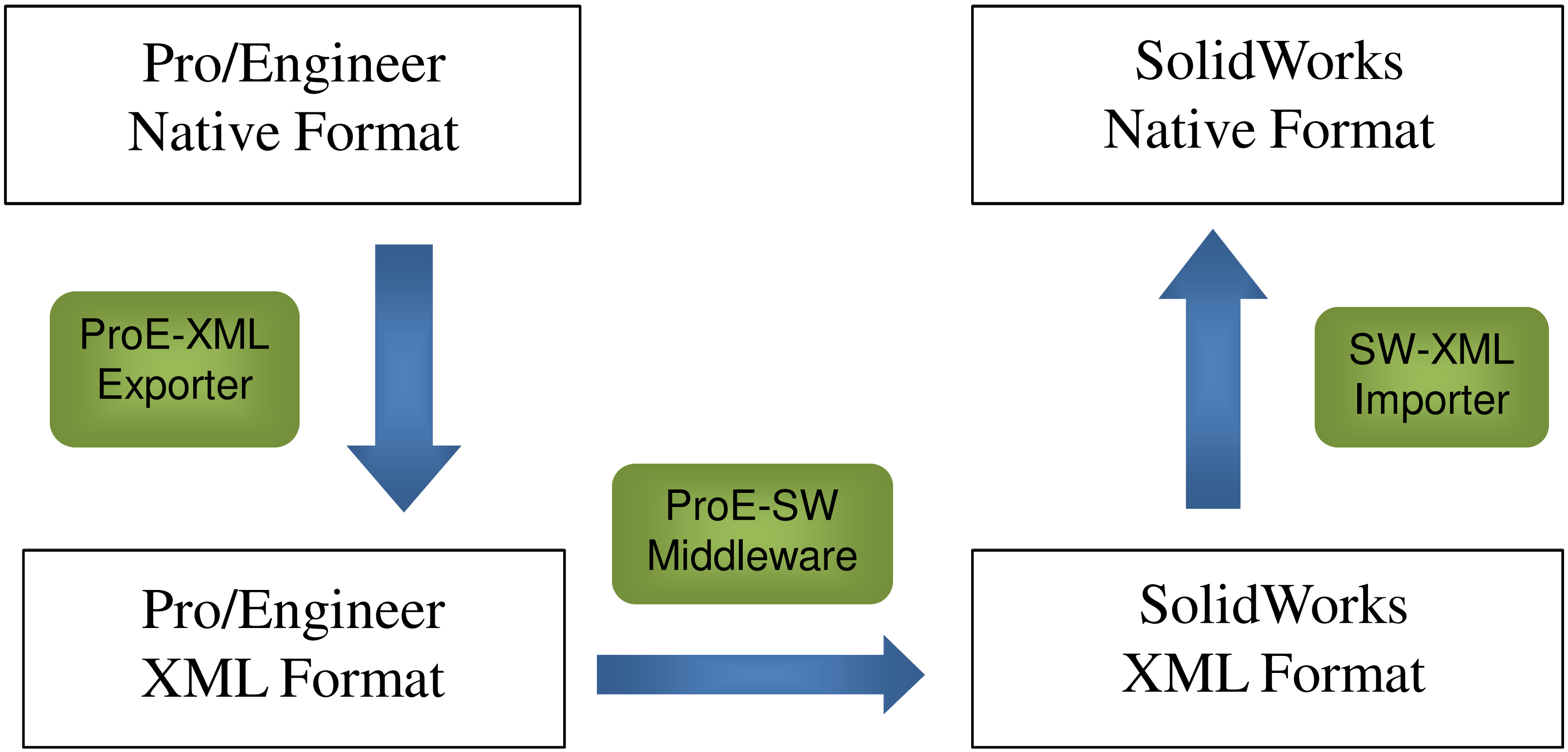




Conversion Strategy

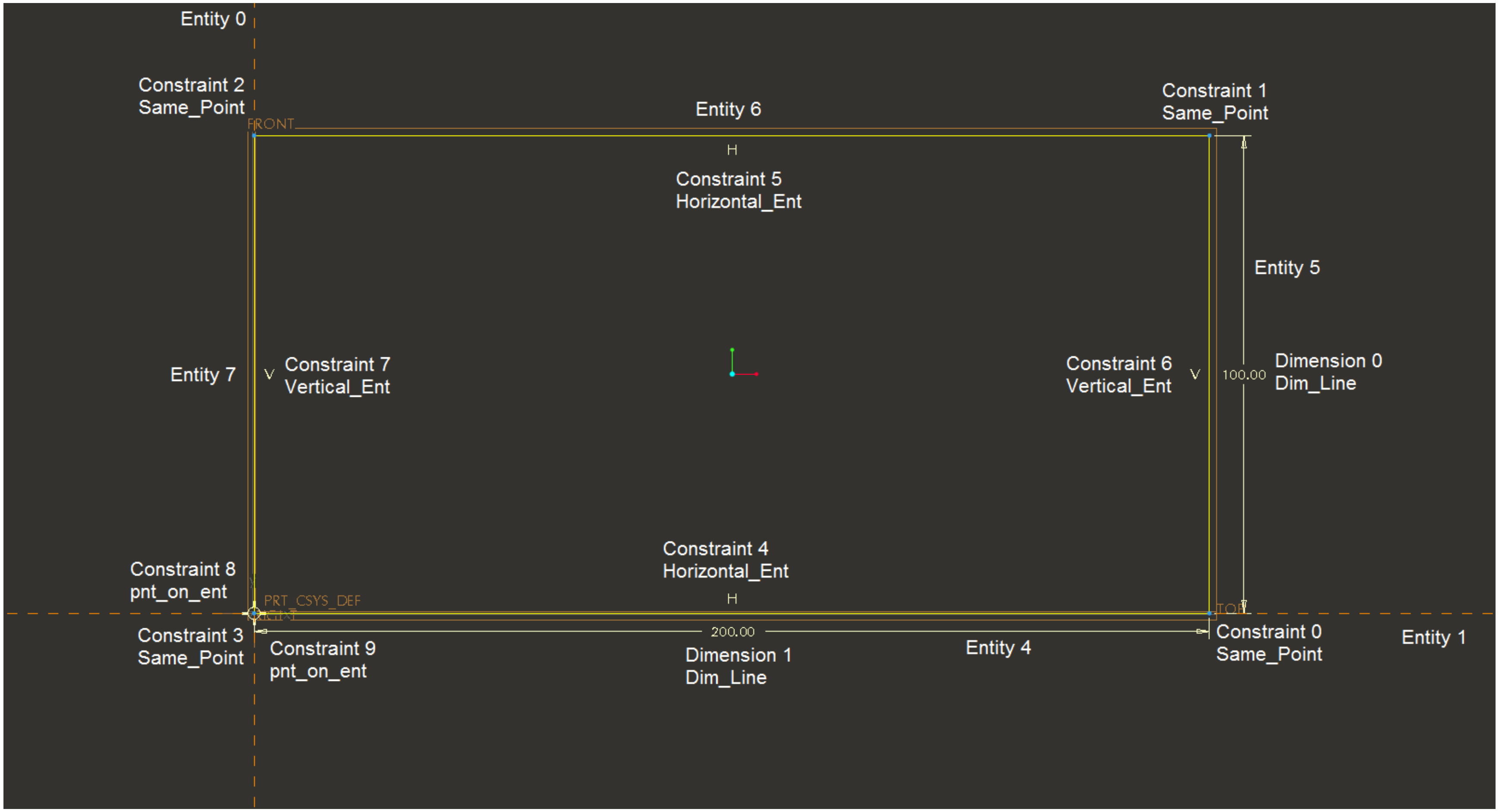


Implemented



- Proof of concept for automatic conversion of 2D sketches.

Pro/E 2D Section



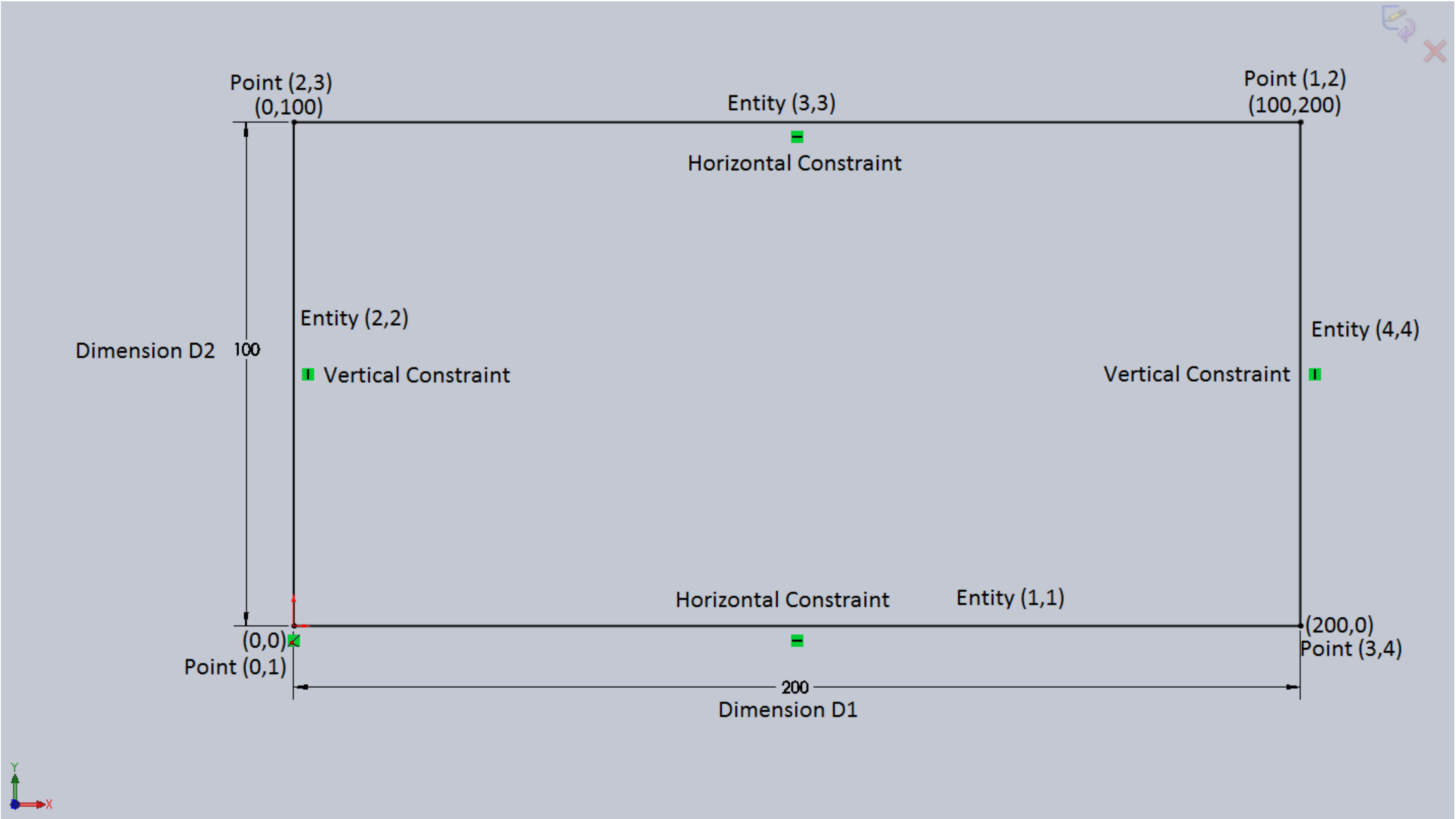
Generated XML of Pro/E Section

```xml
<pro2dsection name="S2D0001">
  <pro2dEntities>
    <pro2dEntity id="4" isProjection="false" type="PRO_2D_LINE" >
      <end1><Pro2dPnt x="0.00" y="0.00" /></end1>
      <end2><Pro2dPnt x="200.00" y="0.00" /></end2>
    </pro2dEntity>
    <pro2dEntity id="5" isProjection="false" type="PRO_2D_LINE" >
      <end1><Pro2dPnt x="200.00" y="0.00" /></end1>
      <end2><Pro2dPnt x="200.00" y="100.00" /></end2>
    </pro2dEntity><!-- ... -->
  </pro2dEntities>
  <pro2dConstraints><!-- ... --></pro2dConstraints>
  <pro2dDimensions><!-- ... --></pro2dDimensions>
</pro2dsection>
```

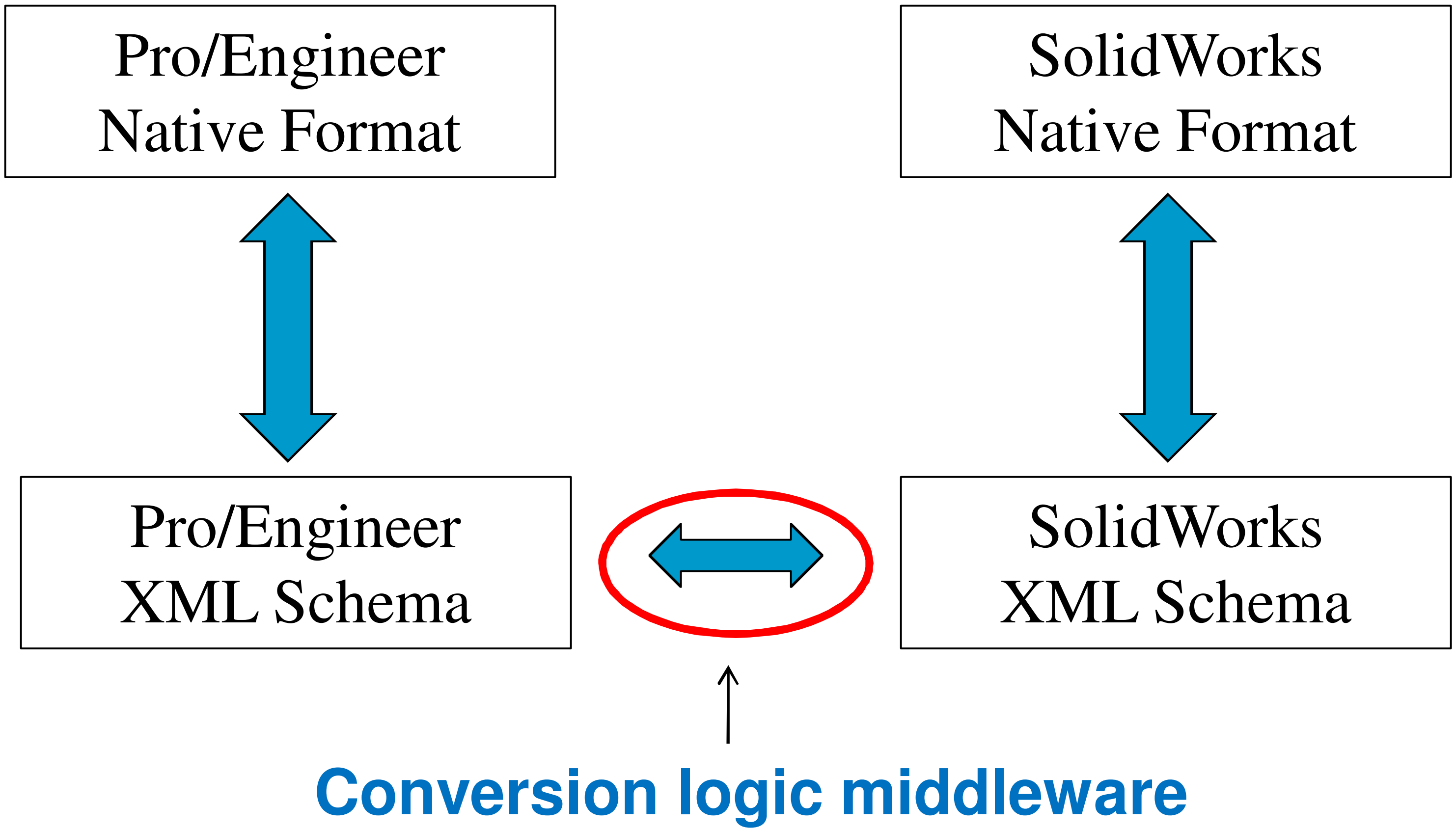
Generated XML of SW Section

```xml
<sw2DSection name="S2D0001">
  <sw2DEntities>
    <sw2DEntity ID="(4,0)" type="swSketchLINE">
      <Start><sw2DPt ID="(4,1)" x ="0.0" y ="0.0" z ="0.0" /></Start>
      <End><sw2DPt ID="(4,2)" x ="200.0" y ="0.0" z ="0.0" /></End>
    </sw2DEntity>
    <sw2DEntity ID="(5,0)" type="swSketchLINE">
      <Start><sw2DPt ID="(5,1)" x ="200.0" y ="0.0" z ="0.0" /></Start>
      <End><sw2DPt ID="(5,2)" x ="200.0" y ="100.0" z ="0.0"/></End>
    </sw2DEntity>
    <!-- ... -->
  </sw2DEntities>
  <sw2DConstraints><!-- ... --></sw2DConstraints>
  <sw2DDimensions><!-- ... --></sw2DDimensions>
</sw2DSection>
```

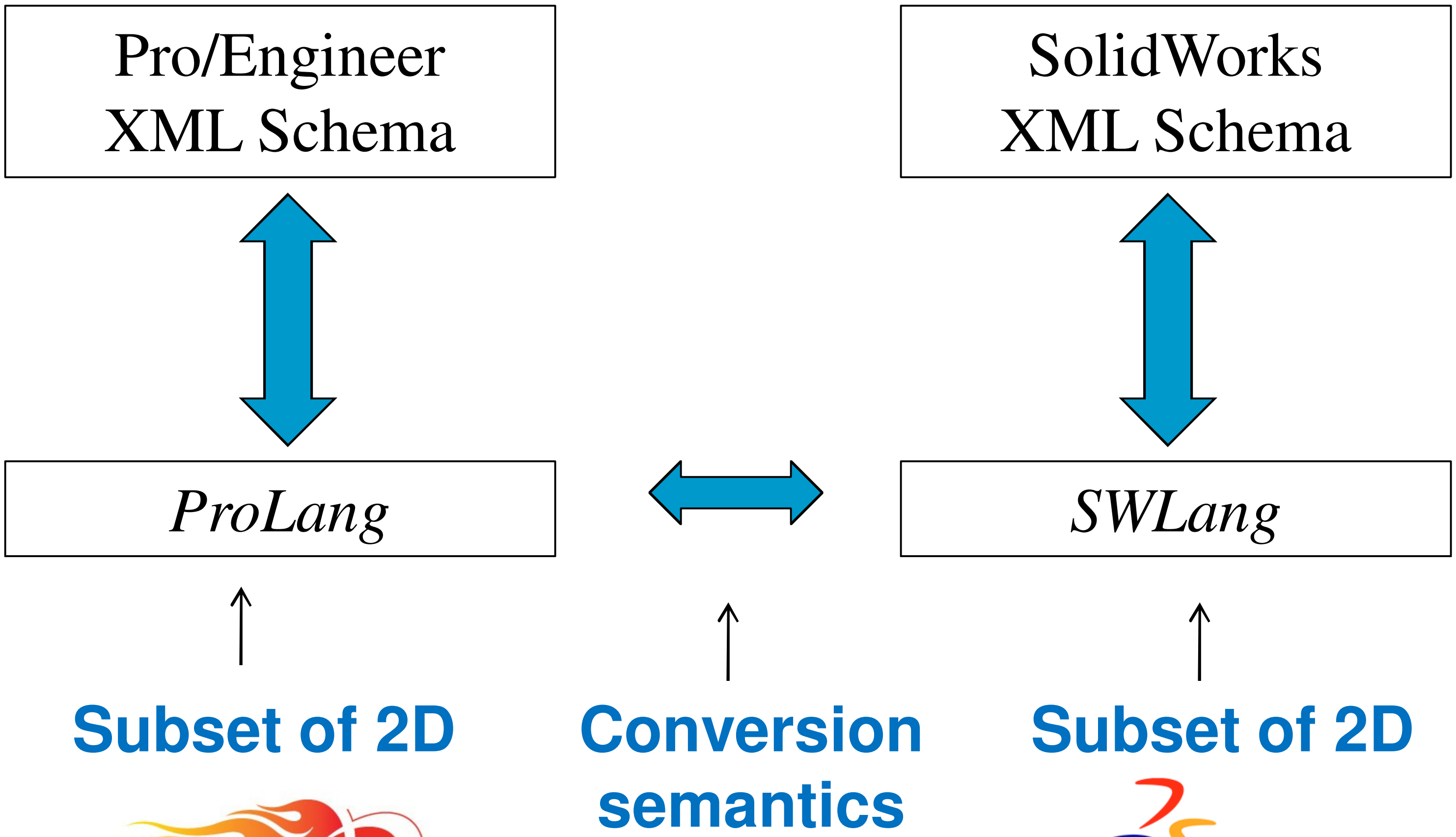
SW 2D Section from XML



Conversion Strategy



Conversion Logic Middleware



ProLang Syntax (Fragment)

Category			AST Node
$Section^P$	S^P	$::=$	$\texttt{Pro2D}(\overline{E^P},\ \overline{C^P},\ \overline{D^P})$
$Entity^P$	E^P	$::=$	$L^P \mid Q^P$
$Line^P$	L^P	$::=$	$\texttt{Line}^P(id^P,\ b,\ P_1^P,\ P_2^P)$
$EntityPoint$	Q^P	$::=$	$\texttt{EntPoint}(id^P,\ P^P)$
$SimplePoint^P$	P^P	$::=$	$\texttt{Point}^P(r_x,\ r_y,\ id^P)$
$Constraint^P$	C^P	$::=$	$\texttt{SamePoint}^P(id^P,\ P_1^P,\ P_2^P)$
		$\mid$	$\texttt{PntOnEnt}^P(id^P,\ L^P,\ P^P)$
$Dimension^P$	D^P	$::=$	$\texttt{LineDim}^P(id^P,\ r,\ L^P)$
		$\mid$	$\texttt{LinePointDim}^P(id^P,\ r,\ L^P,\ P^P)$

$\bar{A} = [A_1, A_2, \ldots, A_n]$, where $n \geq 0$.

SWLang Syntax (Fragment)

Category			AST Node
$Section^S$	S^S	::=	$\texttt{SW2D}(\overline{E^S},\ \overline{C^S},\ \overline{D^S})$
$Entity^S$	E^S	::=	$L^S \mid P^S$
$Line^S$	L^S	::=	$\texttt{Line}^S(ide,\ P_1^S,\ P_2^S)$
$Point^S$	P^S	::=	$\texttt{Point}^S(ide,\ r_x,\ r_y,\ r_z)$
$Constraint^S$	C^S	::=	$\texttt{Coincident}^S(P_1^S,\ P_2^S)$
		|	$\texttt{HorizontalConstraint}^S(L^S)$
$Dimension^S$	D^S	::=	$\texttt{LineDim}^S(idd,\ r,\ E_1^S,\ E_2^S)$
		|	$\texttt{HorLineDim}^S(idd,\ r,\ E_1^S,\ E_2^S)$

Example Differences

- Points in *ProLang* and *SWLang* have 2 and 3 coordinates, respectively.
  - `Point`$^{P}(r_x, r_y)$
  - `Point`$^{S}(ide, r_x, r_y, r_z)$
- *ProLang* constraint `PntOnEnt` has no equivalent in *SWLang*.
- We converted a *ProLang* 2D section containing `PntOnEnt` constraints to an equivalent *SWLang* 2D section.

Conversion Semantics – Transition System

- Conversion semantics is a **transition system** on a set of **states**.
- A state is a pair: (S^P, S^S)
- Transitions have designated **starting states**.

$$(\texttt{Pro2D}(\overline{E^P},\ \overline{C^P},\ \overline{D^P}),\ \texttt{SW2D}([\ ],\ [\ ],\ [\ ]))$$

- Transitions systems complete when they reach one of their designated **final states**.

$$(\texttt{Pro2D}([\ ],\ [\ ],\ [\ ]),\ \texttt{SW2D}(\overline{E^S},\ \overline{C^S},\ \overline{D^S}))$$

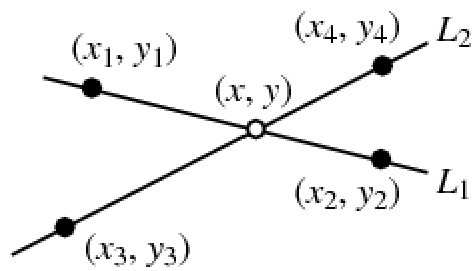
Key Transition Rule

$$\frac{S_2^P \subseteq S_1^P \qquad S_2^P \equiv S_2^S}{(S_1^P,\ S_1^S) \mapsto (S_1^P - S_2^P,\ S_1^S + S_2^S)}$$

- **Premise $S^P_2 \subseteq S^P_1$ is needed to not introduce "new stuff."**

Many-to-Many Mapping

$$\frac{intersect(L_1^P, L_2^P) = P_{int}^P}{\begin{array}{l}\texttt{Pro2D([], [PntOnEnt}^P(L_1^P,\ P^P\texttt{), PntOnEnt}^P(L_2^P,\ P^P\texttt{)], [])} \\ \equiv \texttt{SW2D([}P^S\texttt{], [Coincident}^S(conv_O(P_{int}^P),\ conv_O(P^P)\texttt{)], [])}\end{array}}$$



- The *intersect* function is a partial function that tries to compute the intersection point of two lines.

Summary – Theoretical Contributions

- We presented a rigorous model of CAD systems as programming languages.
- Formally defined algorithm for converting parametric CAD data between CAD systems: Pro/E and SolidWorks.
- We can convert some Pro/E 2D sections containing elements with **no direct counterpart** in SolidWorks.

Summary – Practical Contributions

- **Software** that automatically converts some Pro/E 2D sections to SolidWorks 2D sections.
- Open, text (XML) formats that allow users to write CAD models in native CAD systems' formats **without using the GUI** of CAD systems.
- Other CAD interoperability researchers can apply their logic over our **open, easy-to-parse** formats to experiment with their approaches.
  - No need to learn CAD APIs.

Thank you!