LOCKHEED MARTIN

Envision the Future White Paper Series

A Vision for CAD

A White Paper Presented by:
Lockheed Martin Corporation

Author: Walter W. Wilson
Enterprise Business Services
Fort Worth, Texas

June 2017

I. A Vision for the Future

A. The Problem

What is the best way to represent our CAD (Computer Aided Design) data? Currently we save it in a CAD vendor's secret, proprietary, binary file format. We need a license to access our own data! Our data is effectively held hostage by the CAD vendor! And what about security? Can we be sure a black-box CAD system doesn't contain malware that can subtly steal our design data?

Migrating our data from one commercial CAD system to another is hugely expensive. Millions are being spent to move data from CATIA V4 to V5. Commercial CAD systems and their environments tend to become "obsolete", which encourages/forces this migration. Is it possible to have a representation for our digital data that would not need to be migrated every few decades?

Another problem with commercial systems is their "black-box" nature – the mathematics of the geometry is largely hidden. One can argue that just as engineers working on advanced aerospace projects need deep knowledge of aerodynamics and other sciences, at least some of them should have expertise in the mathematical details of a design. Knowledge and accessibility of the geometric algorithms may be essential for achieving highly optimized designs.

Another recognized issue in CAD is the long-term accessibility of our digital design data (http://www.lotar-international.org/). Design data is supposed to last the life of an aircraft program, which could be decades, perhaps a century for the C-130. Will our CATIA files be readable that long? If the binaries for the CATIA system are somehow preserved, will they be runnable on future computers? (See my "Thoughts on LOTAR" http://www.axiomaticlanguage.org/LOTAR Thoughts.html for more discussion of this problem.)

One candidate solution to the problem of long-term data preservation is STEP (the STandard for the Exchange of Product model data). But STEP has limitations. Some useful geometry types and high-level features are missing. The standard inherently lags behind commercial CAD systems. Information is usually lost when writing to STEP. There is no programmability. Our customizations of the CAD system cannot be saved.

B. The Solution

My solution to the problem of commercial CAD systems has three parts:

1. An engineering design language – I propose that our engineering design data be represented in a textual, human-readable language instead of a CAD vendor's secret, proprietary, binary file format. The language would store high-level definitions of geometry instead of megabytes of numbers. Engineering knowledge could be captured in the definitions. The screen image, drawings, manufacturing instructions, and other data for downstream applications would be generated from the text definitions. History and associativity of the design data would be inherent in the definitions. Parametric design would be encouraged. Programmability in the language would support design automation and optimization and provide for easy customization. Design reuse would be easier. Our designs would be completely independent of a CAD vendor ("platform agnostic").

Examples of existing CAD scripting languages include AutoLISP and PLaSM (http://www.plasm.net/). Text definitions could be typed as text or could be created and edited using interactive graphical operations. (See Sketch-n-Sketch (http://ravichugh.github.io/sketch-n-sketch/) for an example of how this might work.)

2. An open-source geometric engine – I propose that the implementation of this engineering design language – a geometric engine – be open source. My position is that the numerical computations that underlie a design should be accessible to engineers. Our intellectual tools should not be a black box! State-of-the-art projects, like hypersonic aircraft, may require customization of the internal mathematics. Efficiently linking design and analysis tools for design optimization may require low-level source code modifications.

3. Implementation in a minimal declarative language – Finally, I propose that the implementation language for this engineering design language and its geometric engine be a logic programming language called "axiomatic language" (www.axiomaticlanguage.org). The engineering design language would be a domain specific language embedded in axiomatic language. Axiomatic language is a pure specification language, ideal for high-level geometry definitions. Its metalanguage capability would be used to define the features of the engineering design language. Axiomatic language is extremely minimal and elegant, which would make it an ideal long-term standard for the preservation of our design data. (It is orders-of-magnitude smaller than the STEP standard.) Approximate (floating point) arithmetic, for example, would be symbolically defined, which would guarantee identical geometric results on future computers regardless of their floating point hardware.

II. Technical Challenge

The main technical challenge of this vision is that axiomatic language is a pure specification language so its implementation requires automatically transforming specifications to equivalent efficient programs. This program synthesis problem is a grand challenge of computer science. If the implementation target is a parallel machine, it subsumes the grand challenge of automatic parallelization.

The small size and elegance of axiomatic language would make it well-suited to proof. Proof would be used to guarantee equivalence between a user's specification and the generated efficient program. One may also be able to prove assertions about specifications to validate them and this could be more powerful than just testing. Note that industrial-scale proof is yet another grand challenge.

III. Business & Customer Benefits

Representing our engineering design data in a textual language implemented in axiomatic language means our data is no longer held hostage by a commercial CAD vendor. An open source implementation for the engineering design language mitigates the risk of hidden malware that might steal our data. The "openness" of the data representation and algorithms would give engineers complete accessibility and understanding of mathematical details. Our CAD system would no longer be a black box. Customizations of geometric algorithms for advanced aerospace applications would be easier. Higher-quality algorithms and better geometry than is currently found in CATIA could likely be achieved.

The engineering design language could record higher-level information – engineering knowledge. Instead of needing "data analytics" to extract high-level insights from low-level data, an engineering design language would help support the capture of those high-level insights in the first place.

Powerful parameterization and scripting could be used to define families of parts. A single script might be able to define a family of UAVs that could be automatically manufactured using 3D printing.

A well-designed data representation would be usable for the life of an aircraft program without becoming “obsolete” and requiring expensive “migration”. An open source implementation in an elegant minimal declarative language like axiomatic language would support executability of the data far into the future. Long-term accessibility and reusability of our design data would help support the sustainment of our aircraft programs. Programmability in the language would support adaptability to unforeseen future needs such as new forms of manufacturing. Programmability would also support the linking of design data to other applications.

Axiomatic language may have a general software engineering benefit. Specifications should be smaller, more readable, more reusable, and more correct than algorithmic code. Thus the language may provide greater programmer productivity and software quality. Proof in axiomatic language should contribute to better software reliability. In particular, we may be able to prove that our software satisfies security properties and this could help with our cyber security crisis.

IV. Next Steps

The core technical problem to solve is the automatic transformation of specifications to equivalent efficient programs. My guess is that this problem is about as difficult as driverless cars. But it may be possible to do a proof-of-concept transformation system in a few hundred hours. Such a system would involve working out the key transformation algorithms and data representations along with sufficient knowledge to transform some tiny examples.