

A Future for System Engineering Tools

Steven Jenkins
NASA Jet Propulsion Laboratory
California Institute of Technology
steven.jenkins@jpl.nasa.gov

Purpose and Scope

- To talk about system engineering tools and the the state of their use
- To describe some problems that inhibit development and uptake of tools
- To describe an alternative future for tools and system engineers using free and open-source software (FOSS)
- To illustrate the potential of this future with a system engineering modeling toolkit built with FOSS components
- To find out what you think and how we might work together

Speaker Calibration

- I'm a working system engineer
- This is a presentation of fact and opinion
- These are my personal opinions and not those of my employer
- I'm not predicting the future; I'm advocating a course of action
 - I believe that there are strong forces that favor this approach, but nothing is certain
- I might be wrong
 - This is known to be survivable
- I might be a little provocative to keep it interesting

Terminology

- By *system engineering* I mean essentially the process described by INCOSE or Buede
 - Building complementary hierarchical views of a complex system under design
 - Physical, Functional, Operational, others
 - Capturing these views in formal documentation
- By *system engineering tool* I mean anything that you use to execute that process
 - Restricted for today to information technology, as the product of system engineering is information
 - With special attention to tools marketed specifically for system engineering

Varying Tool “Strengths”

- Simple drawing tools
 - Examples: Power Point, Visio
 - Help you express yourself clearly
 - Good for making things look exactly right
 - Make you do too much of the work yourself
- Specialized text processing tools
 - Example: DOORS
 - Good for applying structure to documents
 - Too much focus on the document product; not enough on the complementary views in the model
 - The document mindset is hard to dislodge

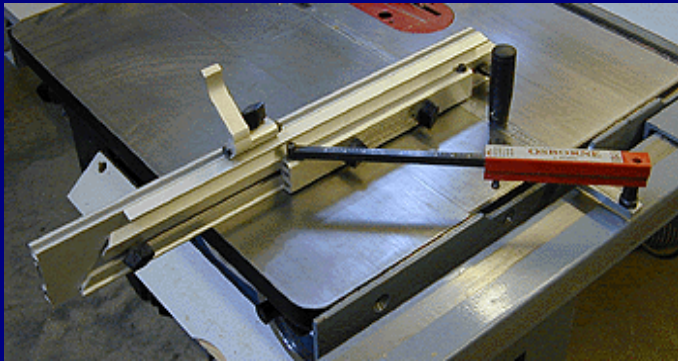
Varying Tool “Strengths”

- “Classical” system engineering tools
 - Examples: CORE, Cradle
 - Support full life cycle
 - Expensive
 - Annoying “security” measures
 - Cumbersome to integrate with other tools
- Software engineering/code generation tools
 - Examples: Rational/Rose, Rhapsody
 - Feature UML, which is Greek to most system engineers, but not necessarily bad
 - Expensive

What's Wrong?

- Imagine being a framing carpenter in this world:
 - The customer provides all hand tools
 - Each house uses a different set of tools
 - Learning one doesn't mean you know the others
 - Jigs you make for one tool won't work for others
 - A jig is a special-purpose tool, often a tool for a tool
- This is a lot like our world:
 - The customer or employer chooses the tool
 - Changing projects or employers often means changing tools
 - Your personal bag of jigs isn't very portable
 - It's hard to accumulate expertise over many years

Example: Table Saw Miter Gauge



- Works with any table saw
 - Use standard interface
- Saves time and money
- Increases your skill
- Learn it once, use it forever

The Price Catch-22

- The market for system engineering tools is small compared to more general purpose tools (e.g., Visio)
- This, among other things, leads to relatively high prices (compare with \$500 for Visio Professional, a highly capable tool in its domain)
- Which keeps the market small, which....
- When the tool is expensive, the vendors waste resources building security features instead of features you care about
 - These “features” get in the way of doing work

An Alternate Future

- Tools are “free” in the sense of being unencumbered
 - They're also low-cost *because* they're unencumbered
- Free system engineering tools create a *lingua franca* for SE the way the web has:
 - How many of you used the terms “home page”, “web site”, “link”, or “blog” before 1990?
- SE tools have a long lifetime
 - Tools become more powerful to meet user needs
 - Users become more powerful as their tools grow
- Is this some kind of Communist fantasy?

Free and Open Source Software

- There has been a significant movement in the software world since 1984 to create a complete software environment (kernel, utilities, applications) free of all proprietary claims
- This movement is *highly* politicized (in ways that do not concern us right now) and not entirely unified
 - The “Free” and “Open Source” camps don't agree
- This movement has produced some things you may have heard of
 - Linux, Perl, Netscape
- And lots of things you probably haven't

Other FOSS Products

- OpenOffice
 - Full-featured office automation suite
- GCC
 - High-performance compiler suite
- KDE, GNOME
 - Desktop environments and toolkits
- MySQL, PostgreSQL
 - Relational DBMS
- Mozilla suite
 - Web browser, email
- Python, Ruby
 - Object-oriented languages
- Apache
 - High-performance web server
- Sendmail, Postfix
 - Email transfer agents
- TeX, LaTeX
 - Typesetting

FOSS Success Stories

- RedHat (Linux vendor) posted 22% 12-mo profit margin ending Nov 2004
- Netcraft says Apache is by far the most popular web server on the Internet
- IBM has made major investments in Linux and open source business
 - And claims to have made it back already
- Amazon claims 25% savings switching to Linux
- MySQL AB has sold commercial support for a free product for more than five years
- Firefox is rapidly growing in popularity
 - Try it!

SourceForge

- SourceForge is one of the primary collaboration sites for open source development
- Some interesting statistics
 - More than 1,000,000 registered users
 - 700 new users added daily
 - 97,000 registered projects
 - 70 new projects added daily
 - 12 million page views daily
 - 1.2 million program downloads daily
- Clearly, something is happening here

What Do We Need For SE?

- Database management
- Data architecture
- Text browsing/editing
- Graphical browsing/editing
- Reasoning
- Simulation
- Document production
- Distributed computing framework
- Access control management
- Neutral import/export
- Application frameworks

FOSS Offerings

- Data management
 - MySQL: full-featured high-performance RDBMS
 - phpMyAdmin: web-based MySQL administration
 - PostgreSQL: robust object-relational DBMS
- Data architecture
 - Protégé ontology editor
- Text browsing/editing
 - Protégé knowledge base editor
- Graphical editing/browsing
 - Dia: Visio work-alike
 - ArgoUML: UML modeling tool suite

FOSS Offerings

- Reasoning
 - CLIPS: expert system tool
 - OpenCyc: knowledge base and reasoning engine
- Simulation
 - Ptolemy II: modeling and simulation framework
 - SimPack: discrete event simulation package
- Document Production
 - SGML/XML/DocBook: semantic markup for technical documentation
 - OpenJade: XML processor
 - TeX/LaTeX: typesetting
 - XML-based web publishing tools

FOSS Offerings

- Distributed computing framework
 - Zope: Web application server
 - Rails: Web application framework
- Access control framework
 - OpenLDAP: online directory
 - GPG: public-key encryption toolkit
- Neutral exchange formats
 - SysML/XMI/STEP
- Application frameworks
 - Eclipse: universal development platform

Focus: Protégé Ontology Editor

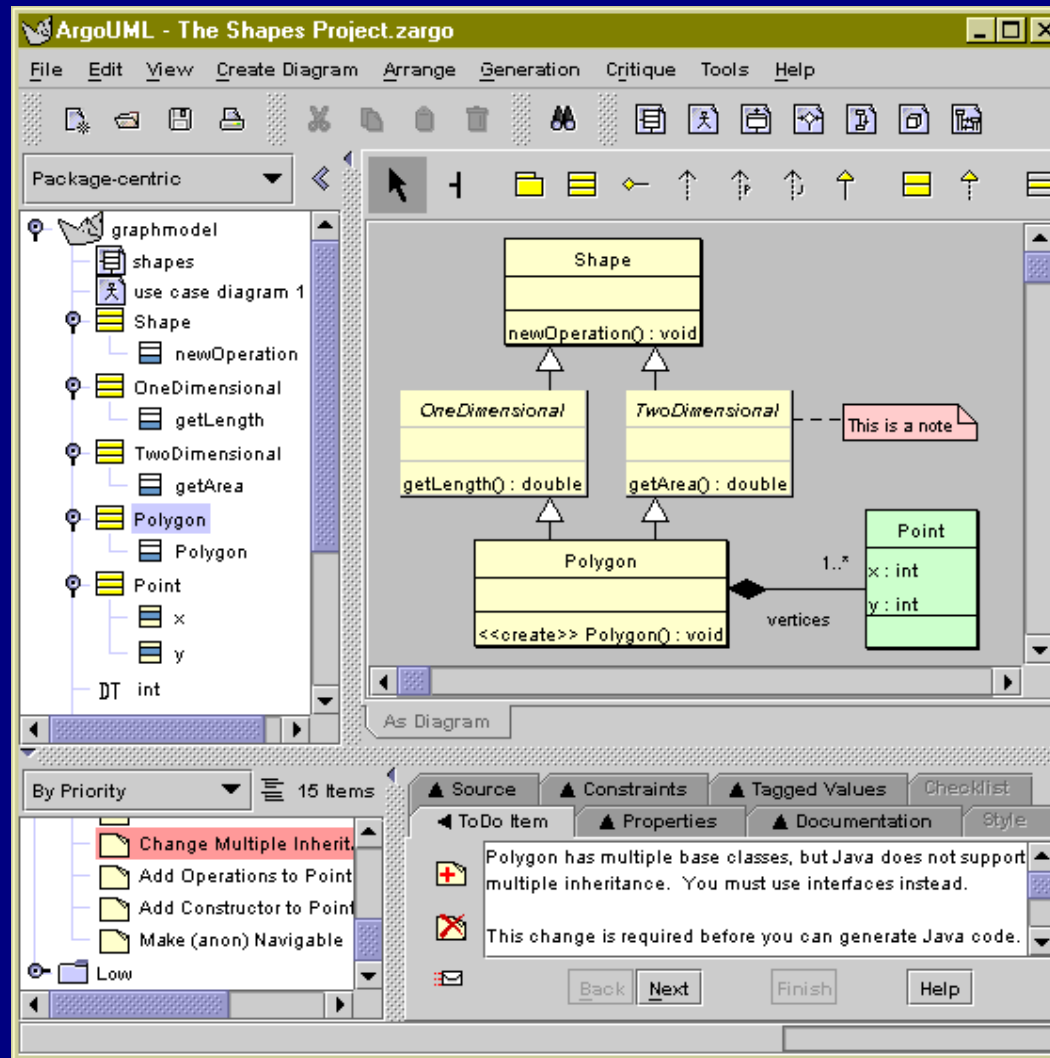
The screenshot displays the Protégé Ontology Editor interface. The top window title is "c4isr Protégé@2.1.2 (file:/home/sjenkins/owl/c4isr.pprj, OWL Files)". The menu bar includes "Project", "Edit", "Window", "OWL", "Wizards", "Code", and "Help". The toolbar contains various icons for file operations and editing.

The main interface is divided into several panes:

- Subclass Relationship / Asserted Hierarchy:** A tree view showing the ontology structure. The root is "owl:Thing", followed by "Element". Under "Element", there are several subclasses, including "AdministrativeElement", "Engineer", "Leader", "Annotation", "NumberedElement", "DecomposableElement", "InformationUnit", "Item", "Product", "ProcessingUnit", "Activity", "Function", "EngineeringElement", "Category", "CompletionCriterion", "Constraint", "Document", "DomainSet", "ExternalFile", "ExternalGraphic", "ExternalText", "Glossary", "Interface", "Issue", "Link", "OriginatingRequirement" (highlighted), "PerformanceIndex", "Program", "Project", "Resource", "ResponsibleOrganization", "Risk", "TestConfiguration", "TestProcedure", "Text", "VerificationEvent", "VerificationRequirement", "WorkPackage", "WorkUnit", "ImplementationUnit", "Component", and "System".
- Class Details:** The central pane shows details for the selected class "OriginatingRequirement (type=owl:Class)". It includes a "Name" field with "OriginatingRequirement", an "rdfs:comment" field, and an "Asserted" tab showing "Asserted Conditions". Under "Asserted Conditions", "EngineeringElement" is listed with a "NECESSARY" constraint.
- Annotations:** A table with columns "Property", "Value", and "Lang".
- Properties:** A list of properties for the class, including "abbreviation (single String)", "annotated_by (multiple Annotation)", "causes (multiple Risk)", "creationStamp (single dateTime)", "creator (single String)", "description (single String)", "generates (multiple Issue)", "modificationStamp (multiple dateTime)", "number (single String)", "owned_by (single Element)", and "uuid (single String)".
- Disjoints:** A section for defining disjoint classes or properties.

The bottom status bar shows the current view is "Logic View" and "Properties View". The system tray at the bottom includes the user name "sjenkins@tidal" and the application path "c4isr Protégé@2.1.2 (file:/home/sjenkins/owl)".

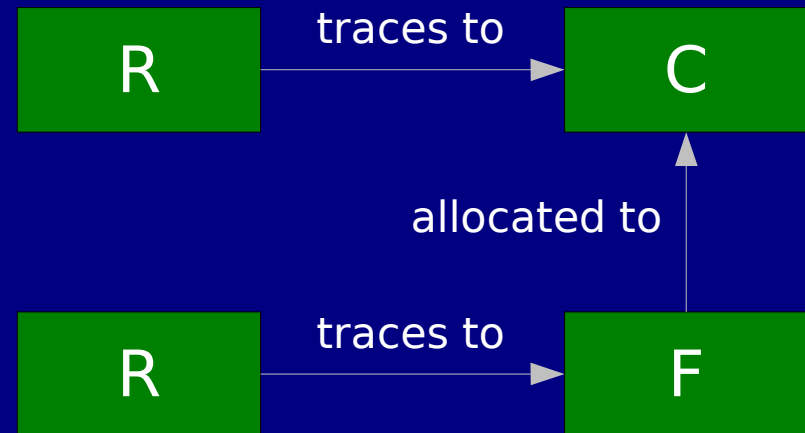
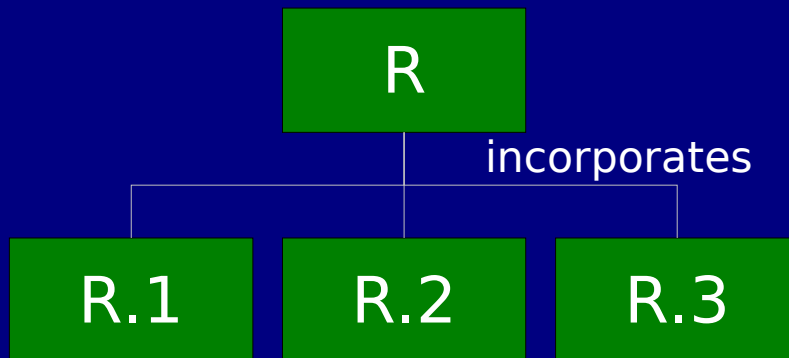
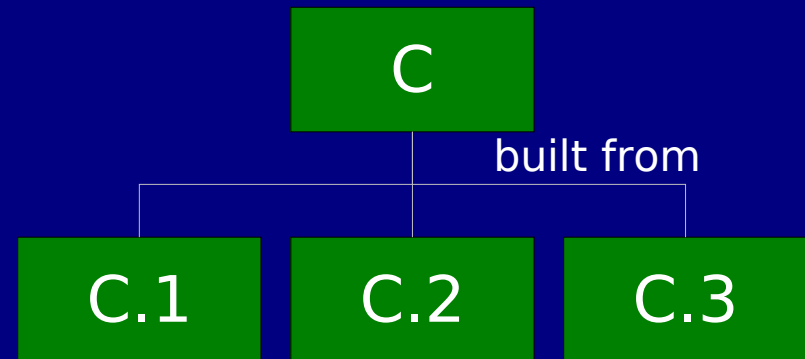
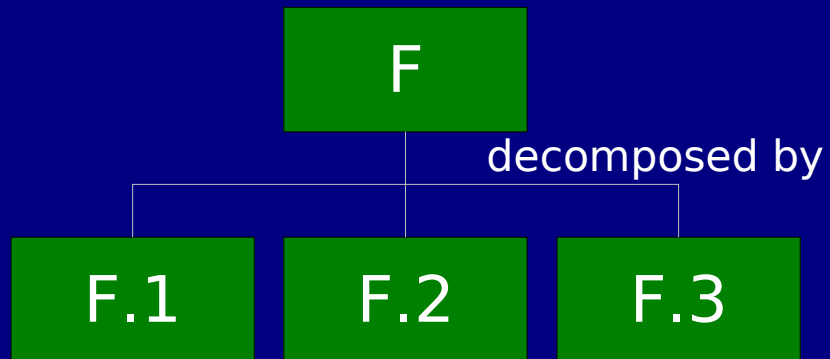
Focus: ArgoUML



A Real Example

- Just to show what you can do with FOSS parts
- A system engineering model application
 - Database consisting of
 - Elements: Components, Functions, Requirements
 - Relationships: 'incorporates', 'traces to', 'allocated to', 'built from', 'decomposed by'
 - Document production system
 - Extracts elements from the database
 - Produces a MIL-STD-490-like document in DocBook format
 - DocBook file can be translated to HTML or high-quality PDF
 - No user interface yet (not my strength)

Simplified Model Schema



Size of the Test Model

- Components
 - 6 levels deep
 - Every component has 3 children except Level 6
 - Total of 364 components
- Functions
 - 3 functions allocated to each component
 - Function n allocated to child component is a child of function n allocated to parent
 - Total of 1092 functions

Structure of the Test Model

- Requirements
 - 5 non-functional requirements trace to each component
 - 2 functional requirements trace to each function
 - Requirements form a tree like Components and Functions
 - Total of 4004 requirements
- Total of 5460 elements
- Relationships
 - Total of 10555 pairwise relationships
- Stored in a MySQL database
- Populated by a driver script written in Ruby

Spec Document Generator

- Objective: for any of the 121 non-leaf-level components, print a specification document
 - In Section 3.2.1, print all non-functional requirements tracing to that component
 - In Section 3.2.2, print the description of each function allocated to that component, and print the functional requirements tracing to each
 - In Section 3.7, for each child component:
 - Print all non-functional requirements
 - For each allocated function, print all functional requirements
- Script produces DocBook (XML) output

Excerpt from the DocBook Output

- Markup shows structure and content, not presentation
- Presentation is added by back-end processing

```
<section role='subsubsection'>
<title role='subsubsection'>Non-Functional Characteristics</title>
<simpara role='empty'></simpara>
<formalpara role='requirement'>
<title role='requirement'>R.1 Requirement R.1 Title</title>
<para>
This is the text of Requirement R.1.
</para>
</formalpara>
<formalpara role='requirement'>
<title role='requirement'>R.2 Requirement R.2 Title</title>
<para>
This is the text of Requirement R.2.
</para>
</formalpara>
<formalpara role='requirement'>
<title role='requirement'>R.3 Requirement R.3 Title</title>
<para>
This is the text of Requirement R.3.
</para>
</formalpara>
<formalpara role='requirement'>
<title role='requirement'>R.4 Requirement R.4 Title</title>
<para>
This is the text of Requirement R.4.
</para>
```

After Output Processing

The screenshot shows a desktop environment with two windows open. The left window is a Mozilla Firefox browser displaying a document titled "System of Systems Specification". The browser's address bar shows the file path: `file:///home/sjenkins/incose/C.F...`. The document content includes a "Table of Contents" with links to sections 1 through 6, and the following sections:

- 1. Scope**
- 2. Applicable Documents**
- 3. Requirements**
 - 3.1. Definition**
 - 3.2. System of Systems Characteristics**
 - 3.2.1. Non-Functional Characteristics**
 - R.1 Requirement R.1 Title.** This is the text of Requirement R.1.
 - R.2 Requirement R.2 Title.** This is the text of Requirement R.2.
 - R.3 Requirement R.3 Title.** This is the text of Requirement R.3.
 - R.4 Requirement R.4 Title.** This is the text of Requirement R.4.
 - R.5 Requirement R.5 Title.** This is the text of Requirement R.5.
 - 3.2.2. Functional Characteristics**
 - 3.2.2.1. F.1 Function F.1 Title**
This is the description of Function F.1.
 - Requirement R.6 Requirement R.6 Title**
This is the text of Requirement R.6.
 - Requirement R.7 Requirement R.7 Title**
This is the text of Requirement R.7.
 - 3.2.2.2. F.2 Function F.2 Title**
This is the description of Function F.2.
 - Requirement R.8 Requirement R.8 Title**
This is the text of Requirement R.8.
 - Requirement R.9 Requirement R.9 Title**
This is the text of Requirement R.9.
 - 3.2.2.3. F.3 Function F.3 Title**
This is the description of Function F.3.

The right window is a PDF viewer displaying the same document content as a PDF file. The PDF viewer shows the same hierarchical structure of sections and requirements as the browser window. The status bar at the bottom of the PDF viewer indicates "2 of 6" pages and a size of "8.5 x 11 in".

Example Summary

- All done on my home Linux system (1 GHz PIII)
- Model data stored in MySQL
- Application scripts and libraries written in Ruby
 - 519 lines of code total
 - A few hours work, mostly learning new tools
- Og object/relational mapping library used to access database from Ruby
- OpenJade used to translate DocBook to HTML
- Custom script used to drive LaTeX to produce PDF
- Time to extract data for any component and generate both outputs: about 10 seconds

What Does This Prove?

- One guy casually messing around on a <\$1k computer with free software can make something that does real work and scales
- From this, I extrapolate that a handful of people working part-time for a year could produce a usable toolkit
 - Largely by writing interface code to glue existing software components together
 - For the most part, the programming is elementary
 - Some specialized knowledge is required for building user interfaces, etc.
- Should we do it?

Envelope Economics

- It is not unusual for a large company to spend \$20k/yr on software maintenance for tools
 - Some spend a lot more
- Suppose 100 companies decided it's in their interest to invest .1 FTE annually on FOSS
 - Having a staff programmer write code and donating it to the tool suite
 - 10 FTEs is more than some tool vendors have
 - Arrangements like this are not unusual
 - Much FOSS code is written by salaried employees with the blessing of their employers

What About Tool Vendors?

- Our purpose is not to put them out of business
- But (personal opinion) \$5k and up for a single-seat license is not sustainable
 - Not when operating systems are \$200 or less
 - Not when powerful applications are \$500 or less
- So how do people make money with FOSS?
 - Services and Consulting
 - Same way they do with commercial software
- System engineering is *hard*
 - People need help to do it
 - It's easier to sell them help if their CIO doesn't have to approve a large software procurement

Recipe for Consulting Success

- Write the definitive how-to book on SE
 - Not a textbook, but a *workbook*
 - Establish yourself as the expert in getting it done
- Include a CD in the back with the FOSS toolkit
- Give away copies of the book at INCOSE meetings
- Your sales pitch:
 - Take the book; it's yours
 - Take the tools; they're yours
 - Your data is always yours
 - Pay me for my skills, expertise, connections, judgment, knowledge, reputation, experience,

Role of Data Exchange Standards

- Data exchange standards are about interoperability and transparency of design
- They don't address (directly) what you can do with design data
 - Just how you move it around
- But data exchange standards are essential for code reuse of the kind we need
- A tool I develop is useful for you if (and only if)
 - The transformation it performs is useful
 - You can get your data in and out of it
- Parsers and emitters for standard formats should be some of the first code written

Possible Next Steps

- Drum up support
- Set up a collaboration site (e.g., SourceForge)
- Establish a not-for-profit foundation to manage development
 - Like Apache, Mozilla, OpenOffice
- Perform system engineering analysis of tool needs and requirements
- Issue Requests for Technology to see
 - Who has what
 - Who's willing to contribute
- Formulate a development roadmap
- Get to work!

What Do You Think?