Verifying Consistency Between Models

Presented at AVICPS 2013

August Schwerdfeger, Hazel Shackleton, and Steve Vestal Adventium Labs
{august.schwerdfeger,hazel.shackleton,steve.vestal}@adventiumlabs.com

Supported by U.S. Army AMRDEC
Bruce Lewis, COTR
bruce.a.lewis.CIV@mail.mil

DISCLAIMER: Reference herein to any specific commercial, private or public products, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement, recommendation, or favoring by the United States Government. The views and opinions expressed herein are strictly those of the authors and do not represent or reflect those of the United States Government. The viewing of the presentation by the Government shall not be used as a basis of advertising."

Adventium™ LABS

Contents

Motivation

Model Consistency Demonstration

- Survey of earlier UAV workflows
- Consistency of solid and avionics models
- Lessons learned

Comments and Future Research

Adventium™ LABS

Goal: Reduce Cost and Schedule

- Reduce development cost, schedule, and risk by early detection of defects that currently remain latent until integration and acceptance testing.

- Reduce total ownership cost by improved cost modeling and optimization methods during requirements engineering and system architecting.

- Pay particular attention to defects associated with interfaces between, and optimizations that span, cyber models and physical models.

Modeling Can Address Known Issues

- Significant development cost, schedule and risk are due to defects introduced in early development phases but not detected until late phases.
  - Develop models in early phases that allow defect detection analyses.
- Significant numbers of defects are associated with interfaces between components and specifications.
  - Analyze to detect inconsistencies between different models.
- Total cost of ownership has been mostly fixed by the time requirements and conceptual architecture phases are completed.
  - Model the system trade space to enable multi-objective multi-disciplinary analysis and optimization.
- There is enormous investment and capability in existing modeling technologies and models.
  - Use a model integration approach to leverage existing capabilities.

See backup slides 29-36 for supporting data.

Adventium™ LABS

Approaches to Multidisciplinary Modeling

- New domain-specific language (Ex: Vanderbilt, Ellidiss, EMF)
  - + Improves targeted engineering task (automation, semantics, analysis, generation).
  - + Can leverage existing tools for specialized back-end analysis.
  - - Requires investment in new language, tool, model development.
  - - Scope limited to a new domain, not a broad multidisciplinary multiphase approach.
- Black-box model integration (Ex: iSight, ModelCenter, OpenMDAO)
  - + Full power of legacy languages, tools, models, technology, investment.
  - + Choice of modeling environments by using organization.
  - - Limited to workflows that use models as design-time functions from parameters to metrics.
- Multi-view gray-box model integration (Ex: UML model synchronization, Intentional Software, FUSED)
  - + Full power of legacy languages, tools, models, technology, investment.
  - + Choice of modeling environments by using organization.
  - + New and more powerful workflow capabilities.
  - - Integration of new modeling environment more complex than black-box integration.
  - - Emerging technology, immature, variability among approaches.

FUSED Multi-View Gray-Box Integration Framework

- Objects in modeling environments are typed using a powerful and extensible type system.
  - dimensions, units, frames of reference, uncertainties, ….
  - multiple inheritance/interfaces/meta-data
  - black-box function signatures
  - timed trajectory data
  - model structure (used in this demo)

- Language extension technology allows a natural division between what SMEs specify in domain-specific models and what system engineers specify in workflows.
  - type data (e.g. units, frames-of-reference) not supported by native language
  - unified support for system configuration and trade space definition
  - facilitates integration of new modeling environments

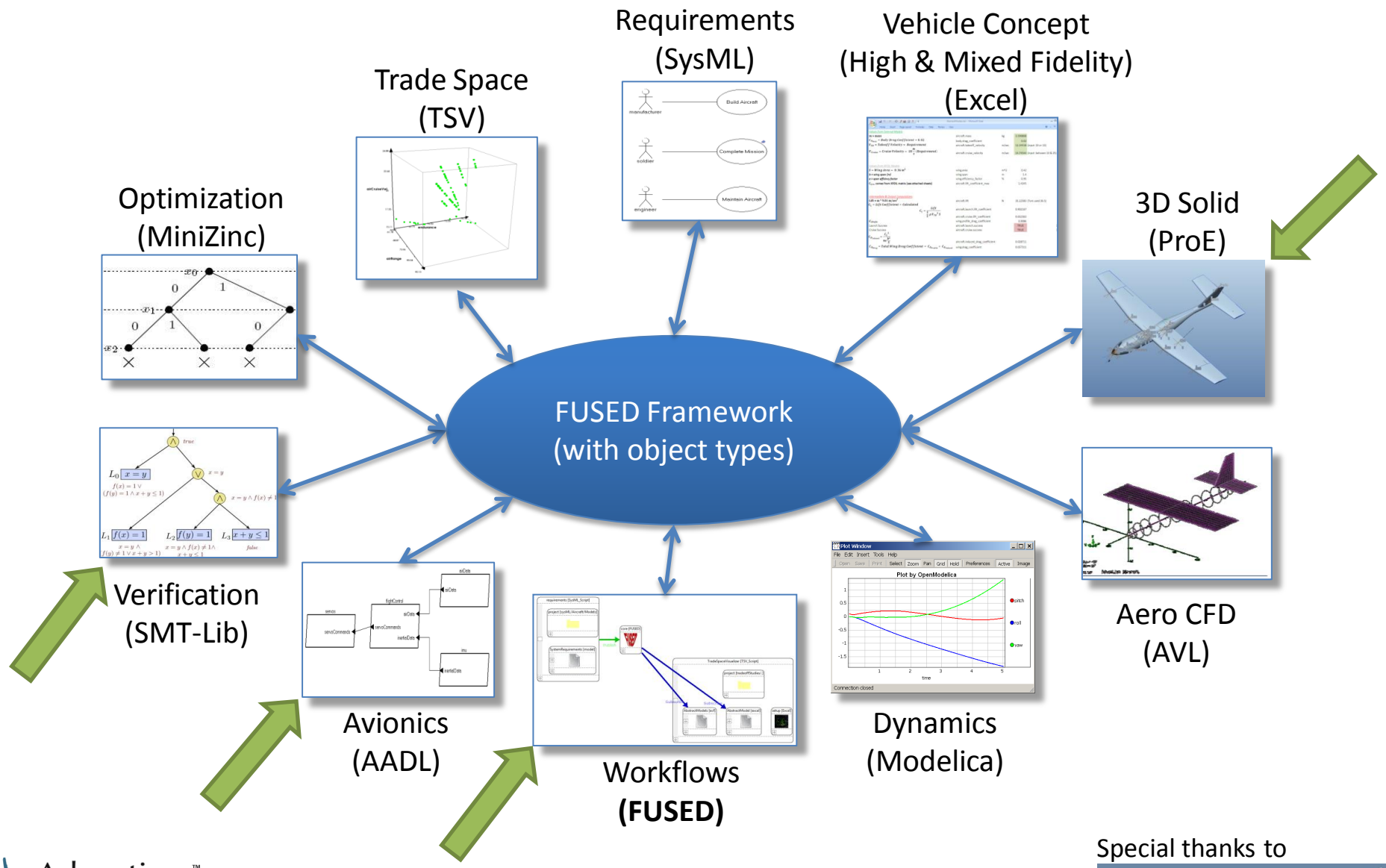
- Workflows beyond data flows between black-box functions are possible.
  - models invoking other models as black-box functions (e.g. TSV)
  - mix-and-match design automation environments (e.g. TSV+MiniZinc for mixed initiative user trade space exploration plus technology optimization).
  - collaborative/distributed simulation composition (e.g. HLA)
  - multi-model verification (this demo)

See backup slides 37-45 for more detail.

Adventium™ LABS

Special thanks to
UNIVERSITY OF MINNESOTA Driven to Discover℠

FUSED Environments from UAV Project



Special thanks to

Contents

Motivation

Model Consistency Demonstration

- Survey of earlier UAV workflows
- Consistency of solid and avionics models
- Lessons learned

Comments and Future Research

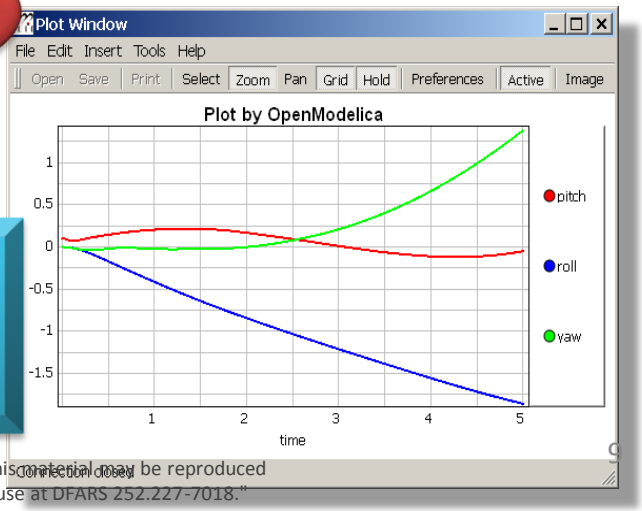
Adventium™ LABS

Solid + Aero CFD + Simulation Workflow

Mass Properties, Center of Gravity, Moments of Inertia, etc.

Stability Derivatives

FUSED



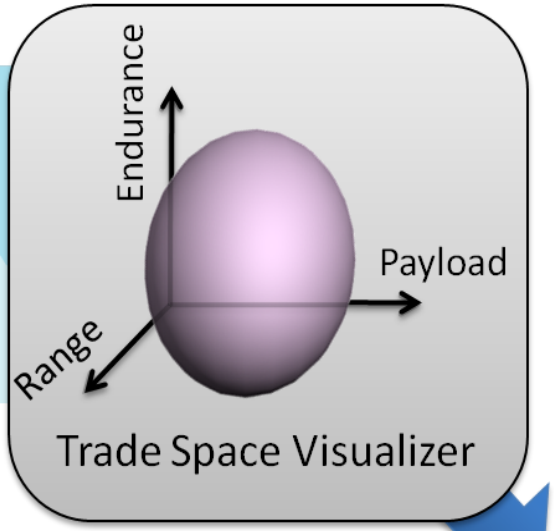
Vehicle Dynamics (Modelica)

- Multi-model configuration
- Eliminate cut-and-paste

Adventium™ LABS

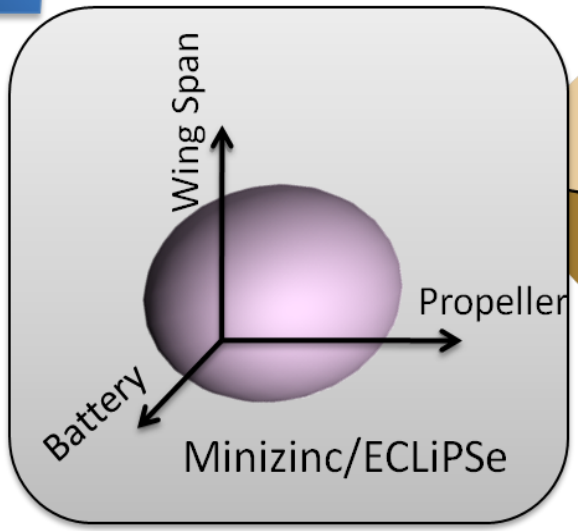
Mixed Initiative Optimization Workflow

Goal Trade-Offs



Customer uses TSV to identify the Pareto frontiers of the available technologies and explore complex trade-offs between requirements.

Design Trade-Offs

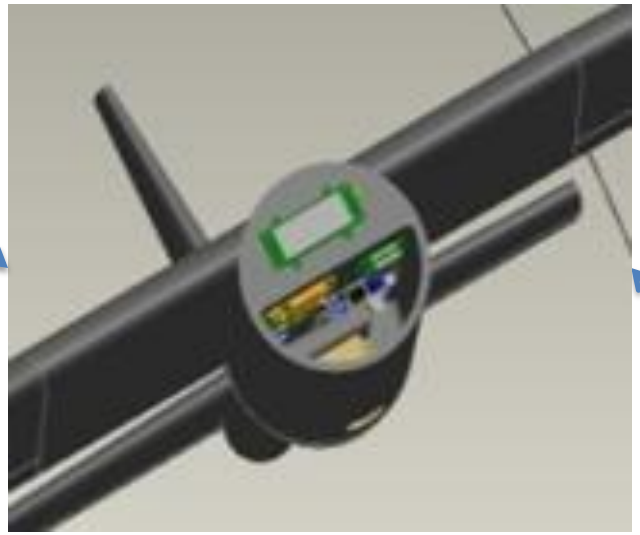


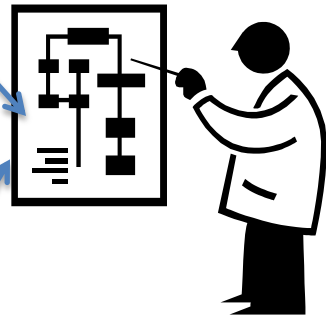
Engineer uses MiniZinc/ECLiPSe to automatically optimize the design for the customer's higher-level trade-off choices.

Adventium™ LABS

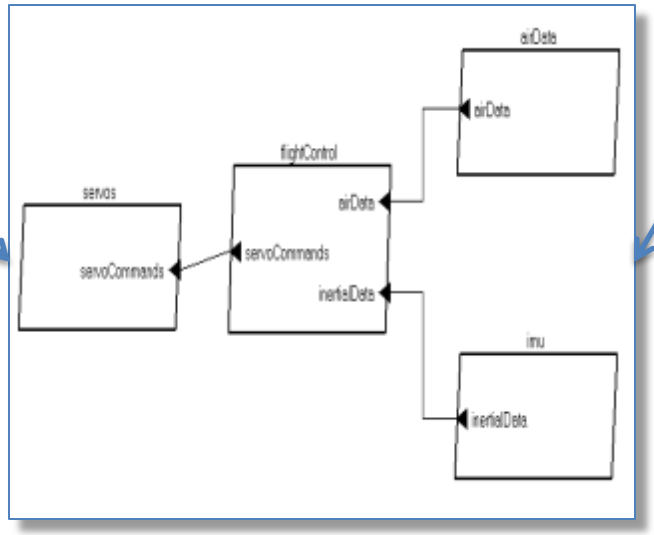
Model Consistency Challenge Problem

Mechanical engineers specify packaging, cable design, physical placement and routing.





Avionics architects specify digital hardware resources, functional connections, and software/hardware allocation.

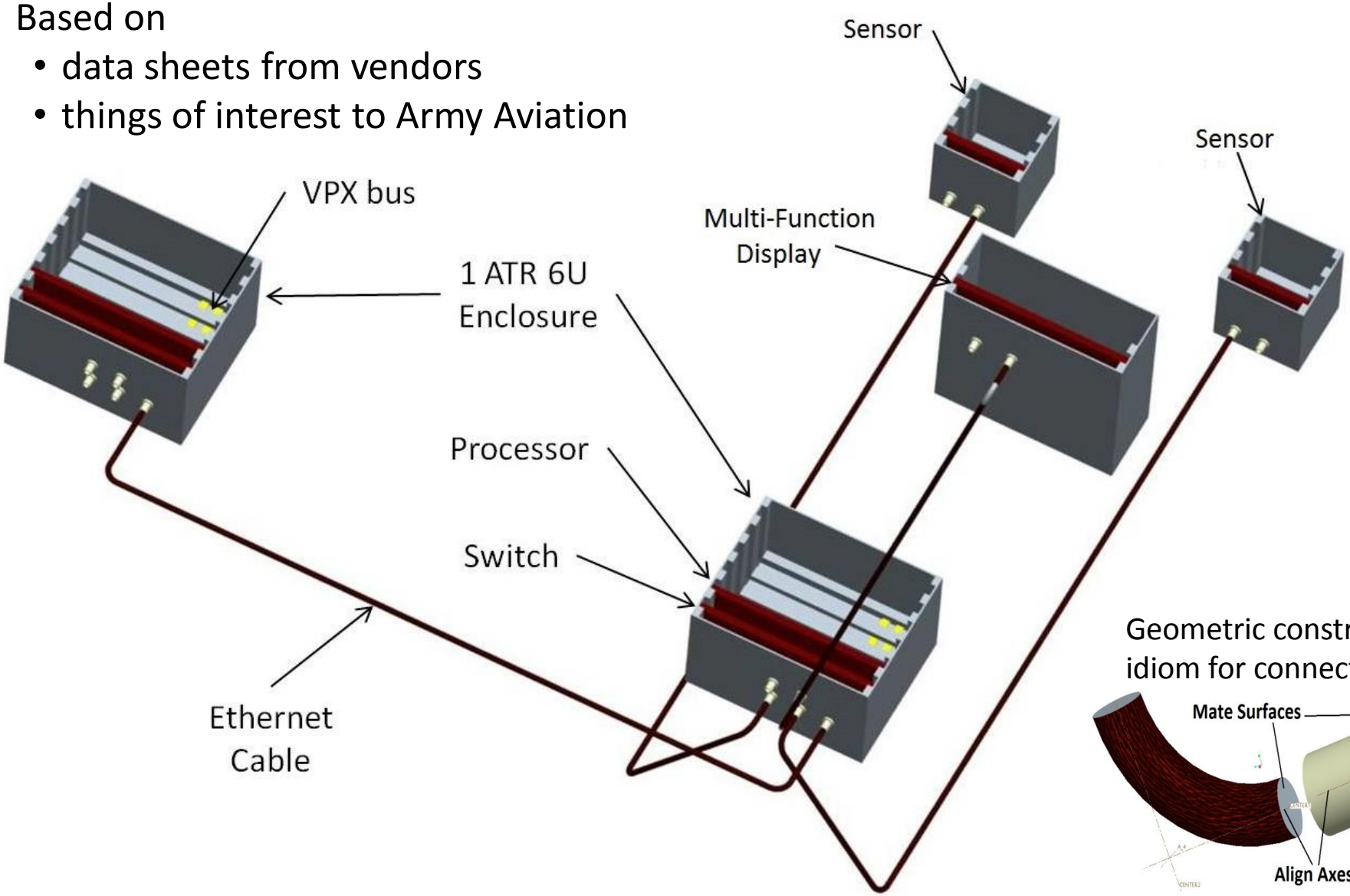


System engineers want assurance the two models are consistent with each other.

Challenge Avionics Solid Model

Based on

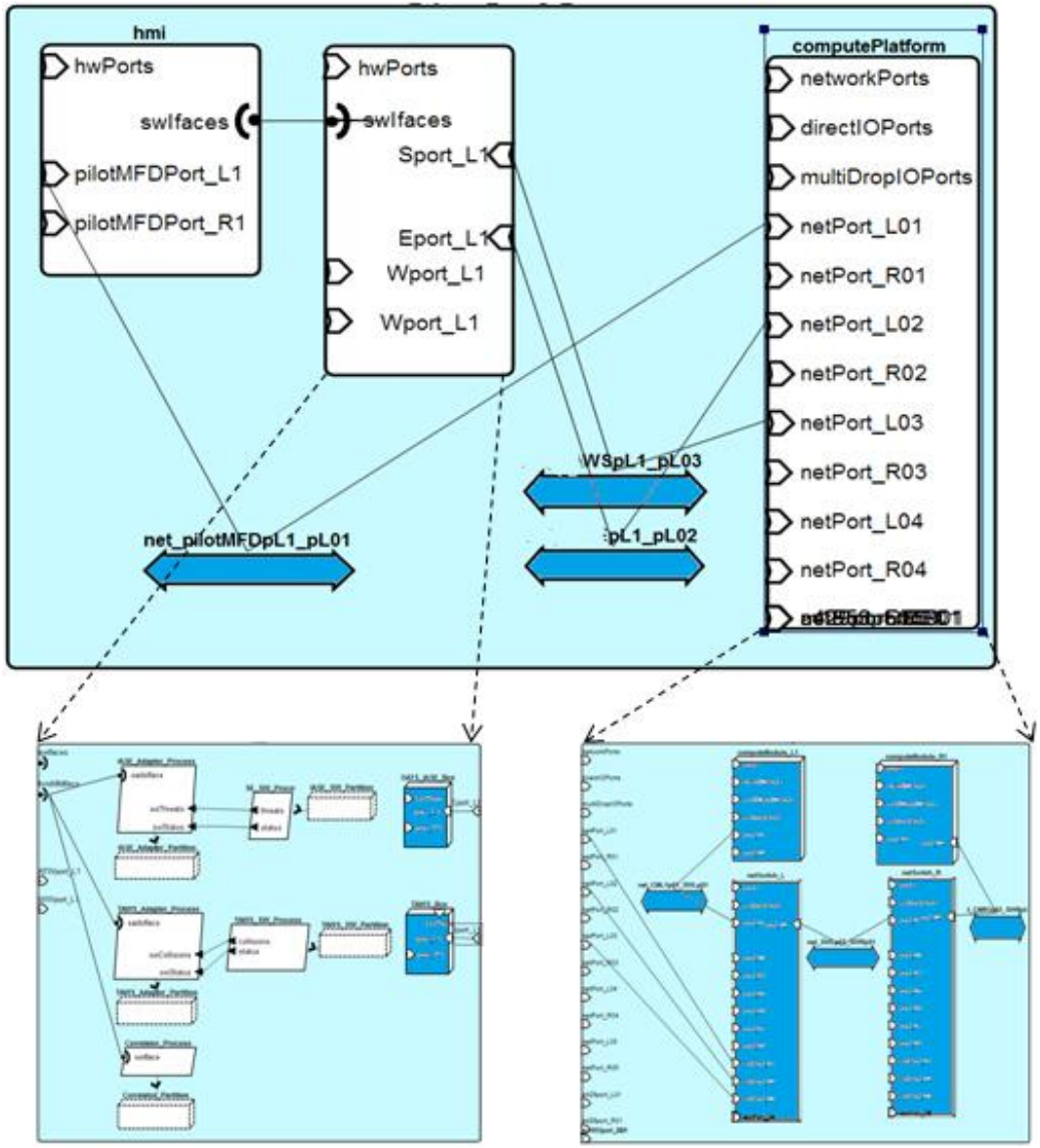
- data sheets from vendors
- things of interest to Army Aviation



Geometric constraint idiom for connections.

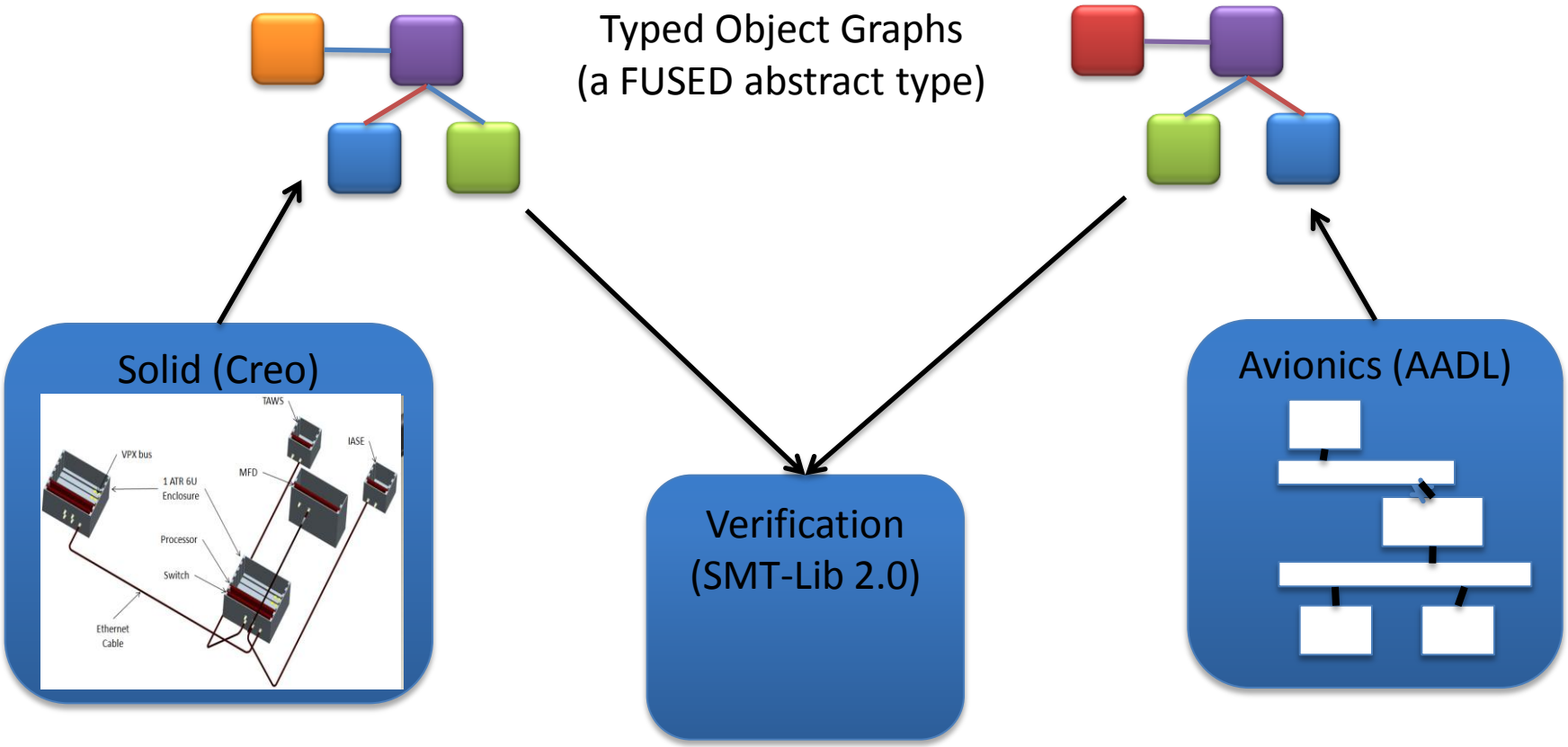
Challenge Avionics Architecture Model

Based on things of interest to Army Aviation



Hardware resources are tinted blue.

Verify Complex Consistency Properties



Verify avionics components and connections of specified types are subgraph isomorphic to the solid model, subject to type compatibility constraints.

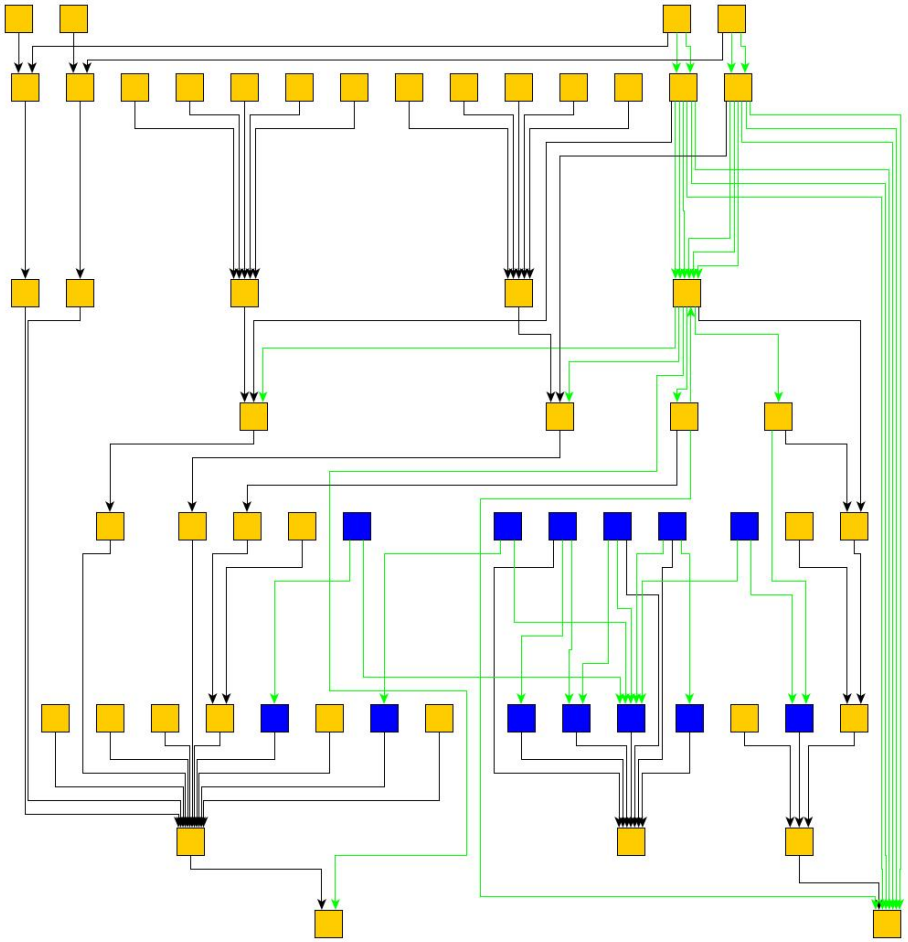
Abstract Graph for AADL Model

hardware nodes

other nodes

connection edges

other edges



More detailed typing information exists but is not illustrated in this diagram.

SMT-Lib Consistency Property

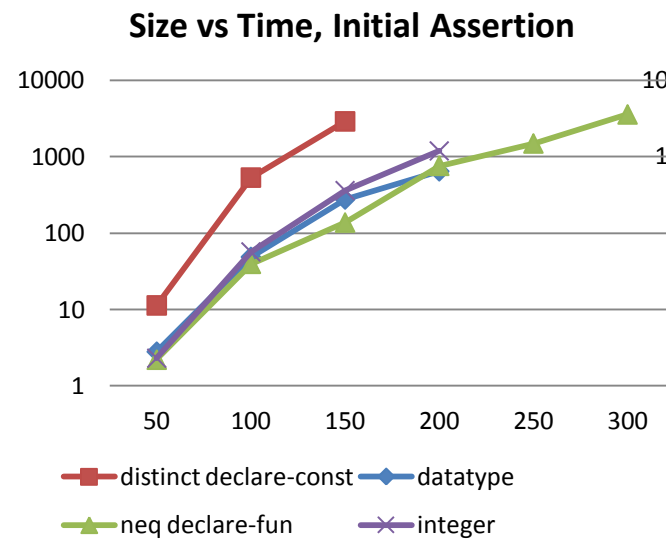
```
(FUSED-subscribe JCAIMA2)
(FUSED-subscribe Mission_System_Design_Instance)

(define-fun avionicsSubgraphObject ((ae Mission_System_Design_Instance.Object)) Bool
  (or (Mission_System_Design_Instance.ofType ae Mission_System_Design_Instance.processor)
   …
))
(define-fun typeCompatible ((ae Mission_System_Design_Instance.Object) (se JCAIMA2.Object)) Bool
   (or (and (Mission_System_Design_Instance.ofType ae Mission_System_Design_Instance.processor) (JCAIMA2.ofType se JCAIMA2.PROCESSOR))
      …
))

(declare-fun map ()(Array Mission_System_Design_Instance.Object JCAIMA2.Object))
; for all objects A in the AADL model that must map to some object A' in the solid model
(assert   (forall ((ae Mission_System_Design_Instance.Object))
 (=>  (avionicsSubgraphObject ae) (and
   ; the map is 1-to-1:  if A maps to A' and B maps to A' then A = B
  (forall ((aeB Mission_System_Design_Instance.Object))
      (=> (and (avionicsSubgraphObject aeB) (= (select map ae) (select map aeB))) (= ae aeB)))
  ; if A maps to A' and A attaches to B and B must map then B maps to B' and A' attaches to B'
  (let ((se (select map ae)))
    (forall ((aeB Mission_System_Design_Instance.Object))
      (=> (and (Mission_System_Design_Instance.attached ae aeB Mission_System_Design_Instance.connection) (avionicsSubgraphObject aeB))
            (JCAIMA2.attached se (select map aeB) JCAIMA2.Mate))))
  ; if A maps to A'  then the type of A is compatible with the type of A'
  (typeCompatible ae (select map ae)))))
)
```

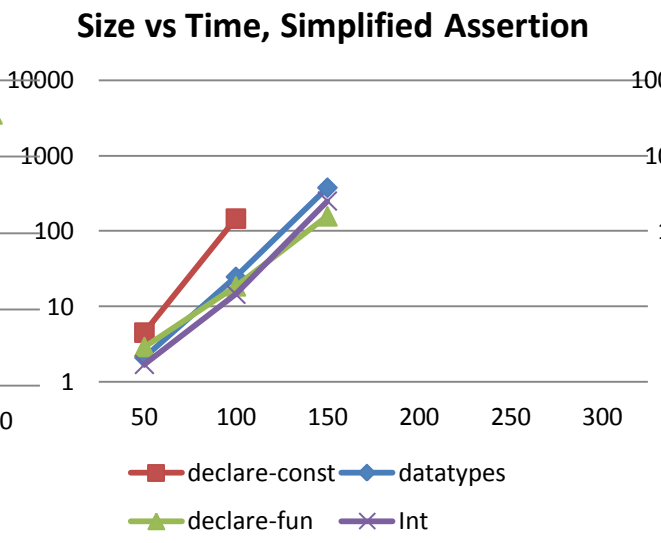
Surprisingly Concise

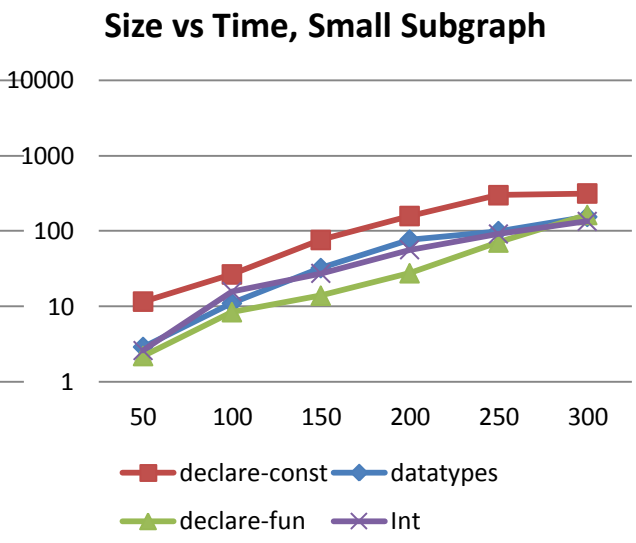
Consistency Verification Benchmarks

Benchmarked alternative formulations:

- distinct declare-const
- datatypes (Z3 extension)
- declare-fun and multiple /= assertions
- integer literals
- simplified sub-graph isomorphism assertion







Adventium LABS

Wiring Harness Lessons Learned

Wiring harness design is a complex domain itself.

- Electrical connection may go through multiple harnesses, e.g. bulkheads, enclosures.
- Solid modeling vendors have special add-in tooling for wiring and plumbing design and electrical properties analysis.
- Redesign and rebuild is not infrequent in practice.
- Electromagnetic/environmental issues may be so complicated that rapid prototyping and lab testing (versus analysis) may be advisable.

Lessons:

- Investigate use of wiring harness electrical analysis outputs from specialty add-in tools as an input for consistency verification (often available in simple CSV format).
- Investigate support for demand-driven “right fidelity” modeling processes that may include rapid prototyping.

Special thanks to
MINNESOTA WIRE Life Saving Connections

SMT Lessons Learned

- SMT-LIB language allowed concise specification of this consistency property (class of properties that there exists a mapping between structural elements of the models that satisfy a set of object and relation first-order predicates).
  - Investigate additional and more complex challenge problems, e.g. consistency between AADL safety analysis and solid zonal common cause analysis.

- Performance and tractability can depend significantly on the combination of model formulation and selected tool.
  - Investigate alternative formulations and multi-backend approaches (e.g. Kansas State LDP, SMT analogue to UMN/RC Lustre framework).
  - Investigate language extension approaches (e.g. UMN MELT) for classes of consistency conditions (e.g. exists a mapping between model structures).

- Clear feed-back on why models are inconsistent doesn't just happen.
  - Investigate application of the above methods to better formulate models and back-translate unsatisfiable core results.

Architecture Modeling Lessons Learned

Consistency properties may depend on modeling idioms (language-specific “coding” guidelines), e.g. different combinations of geometric constraints may be used to attach cables and connectors.

Reference architecture patterns impact consistency (as well as interoperability, portability, etc.), e.g. initial AADL redundancy pattern based on a civil IMA platform was not consistent with initial solid redundancy pattern based on simple OTS packaging.

Lessons:

- When defining modeling guidelines, consider consistency with other models and modeling guidelines.
- Include architectural patterns as part of a reference architecture, e.g. JCA redundancy patterns.

Contents

Motivation

Model Consistency Demonstration

- Survey of earlier UAV workflows
- Consistency of solid and avionics models
- Lessons learned

Comments and Future Research

Adventium™ LABS

Choice of SMT

- There are more efficient verification algorithms for this specific consistency property than using an SMT solver.
- We selected SMT because we are interested in exploring the set of useful consistency properties that might exist and SMT's potential to be a broadly applicable verification technology.
- SMT performance benchmarking was promising, although not what would be needed in practice. Work with an SMT expert is likely to improve performance significantly.
- SMT may still be useful to explore and identify consistency properties for which efficient specialized solvers are worth developing.
- We did not get our tool to provide useful feed-back on why two models were not consistent (why the property was unsatisfiable). It will be necessary to develop this in order for the approach to be useable, even for research.

Broader Research Questions

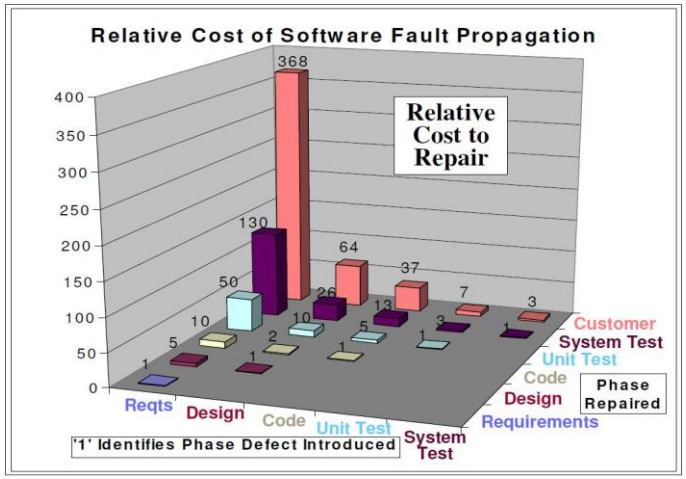
- Our gray-box abstraction was strong typed, but the abstraction relations were not formally specified. Extraction from concrete model to abstract graph was conventional. How can we formally define and verify abstraction relations between gray box views of a model and the concrete model that support a given purpose? How do we do an end-to-end verification of both abstraction operations plus the consistency property over the two abstractions?

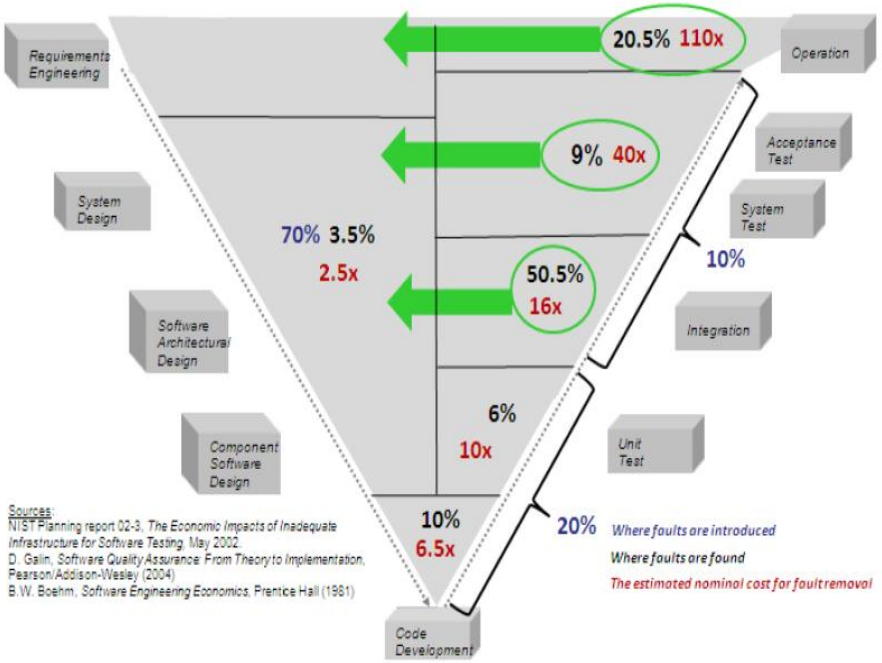
- A goal is improved detection of defects during earlier phases of the project. How effective is a given consistency property at detecting defects that currently remain latent until system integration or acceptance testing or fielding?

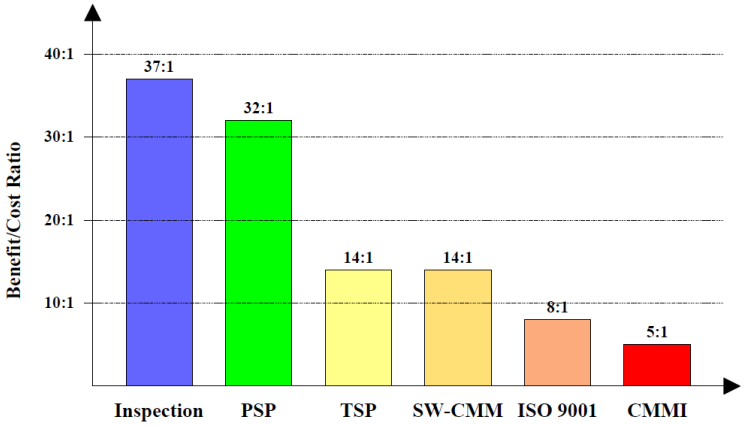
Backup Slides

(see notes for citations)

Expensive Defects are Made Early and Detected Late





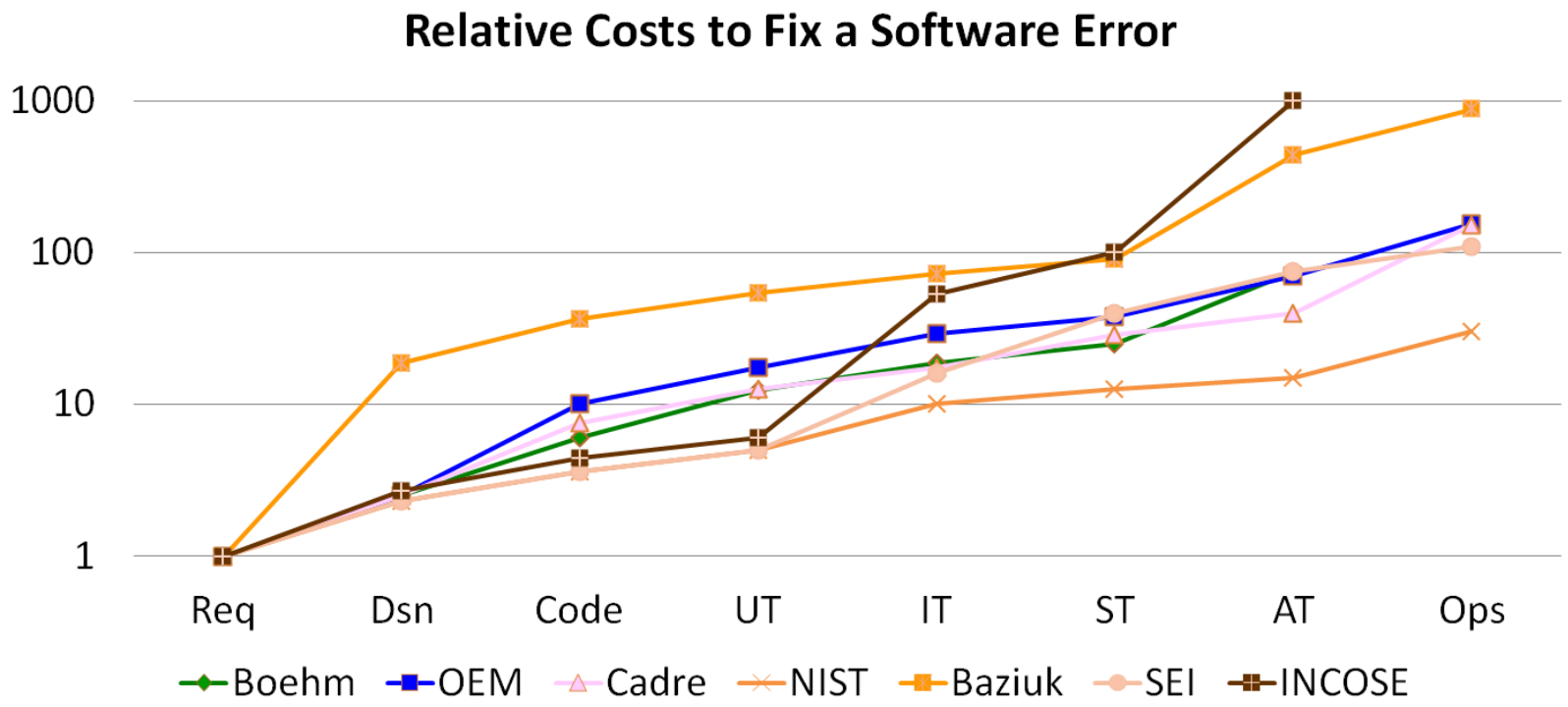


SAVI estimated $400M cost avoidance using virtual (model) integration in early phases.

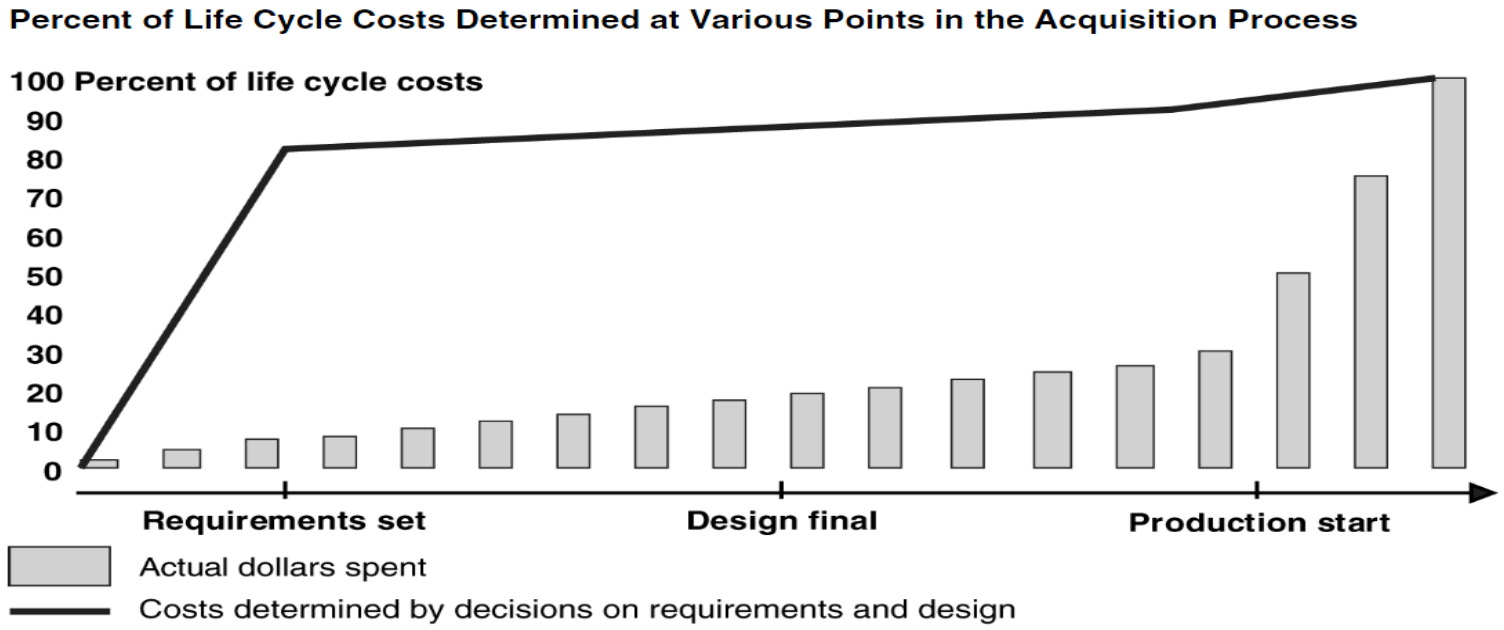
SAVI estimated positive ROI from just 10% improvement in early detection effectiveness

Adventium™ LABS

Survey of Relative Repair Cost



Most Cost is Determined by Requirements



- "Studies have shown that by the time a product is ready for development, over 90 percent of the operating and support costs have been determined." (GAO)
- "Missing predicted MTBUMA goals by just one percent could result in an increase in O&S costs of over $75 million." (Dellert)
- "If we thoughtfully analyzed the FOMs of COST, SCHEDULE AND PERFORMANCE we would always conclude poor reliability is THE dominant cost driver." (Eaton)

Adventium™ LABS

Problems Occur at Interfaces

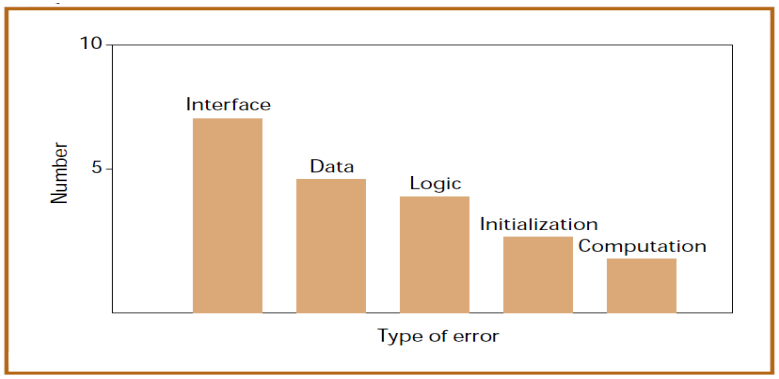
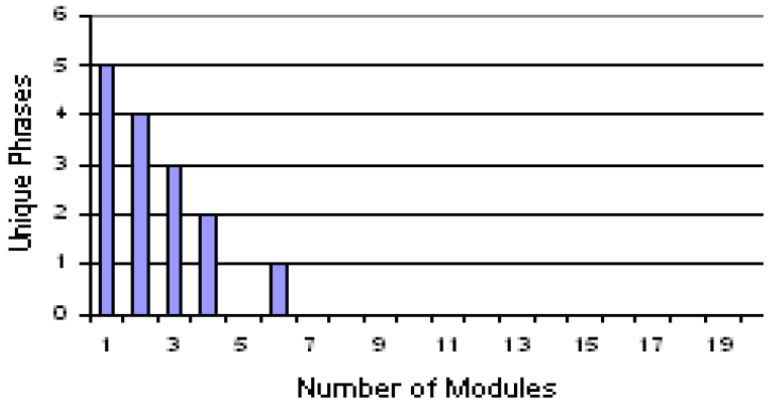


Figure 2. This Pareto chart example shows that Interface defects comprised the largest defect class.

- Between components
- Between development phases
- Between disciplines
- Between organizations



Occurrence of misinterpreted phrases in defect reports.

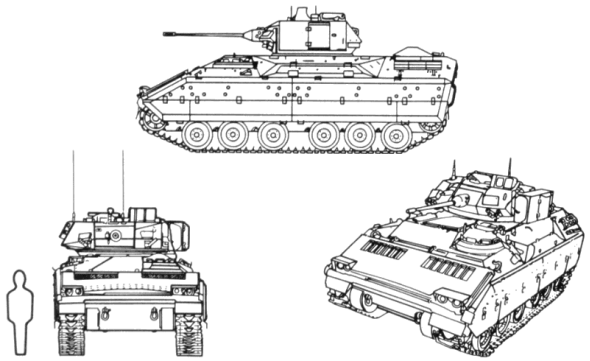
"58-62 percent of failures in our data sets map to faults in more than one file" (Maggie Hamill et al.)

"Safety-related software errors arise most commonly from (1) discrepancies between the documented requirements specifications and the requirements needed for correct functioning of the system and (2) misunderstandings of the software's interface with the rest of the system." (Lutz)

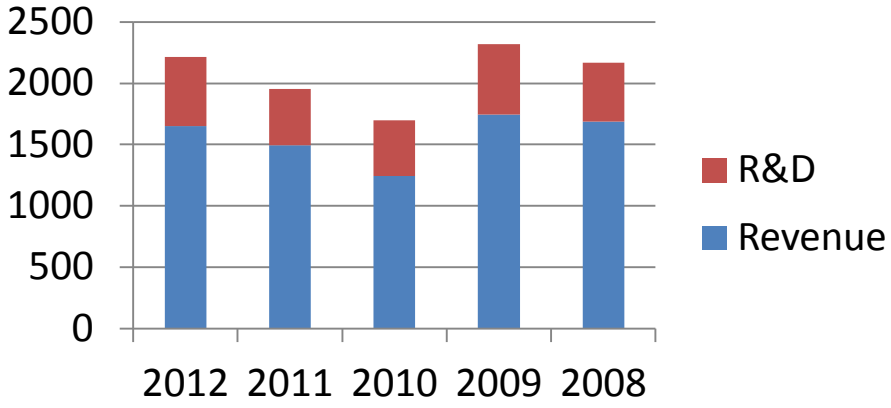
We still lose vehicles due to units mismatch.
(Mars Climate Orbiter 1999)

Adventium™ LABS

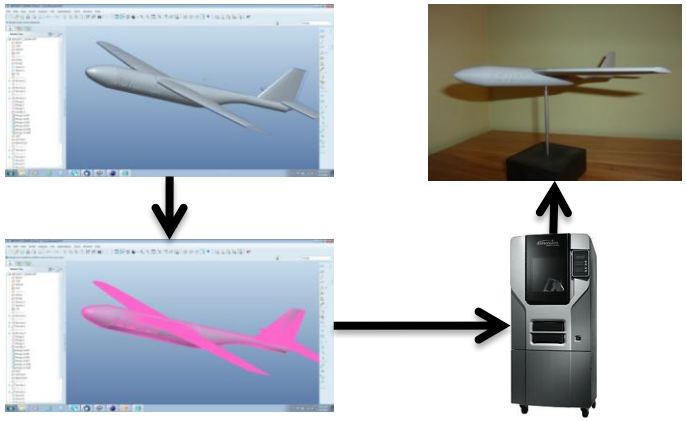
Leverage Existing Modeling Capabilities



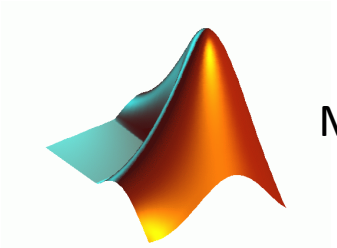
BAE uses about 450 different models for DoD ground vehicle development



AutoDesk Revenue and R&D ($M)



Additive Manufacturing > $1B/year, $3B by 2016, $5B by 2020



MatLab > 1M users

M&SCO estimates $ billions invested in legacy M&S Resource Repositories

Adventium™ LABS

Requirements & Designs are Dynamic

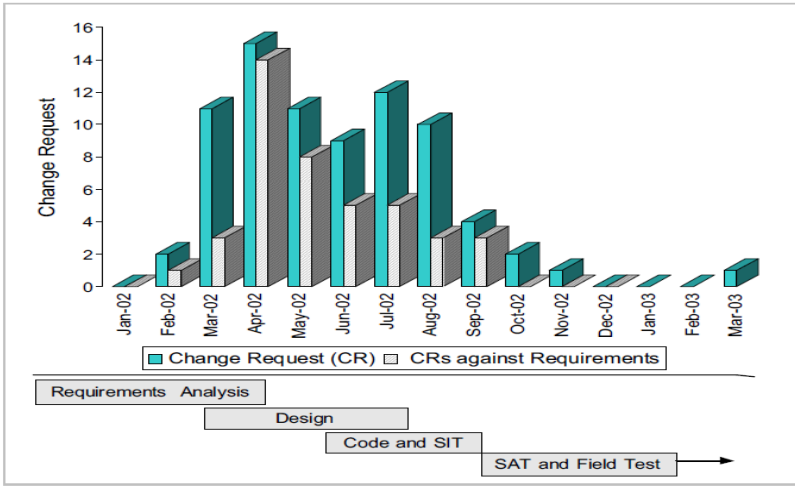


Figure 2. Change requests arrival rate

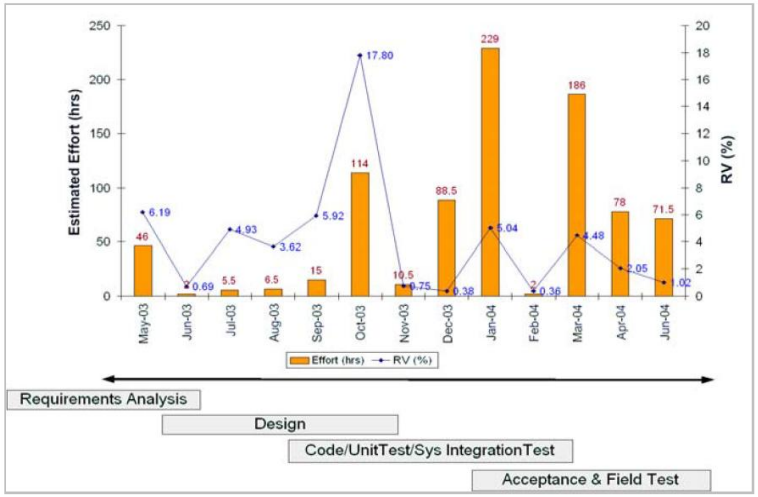
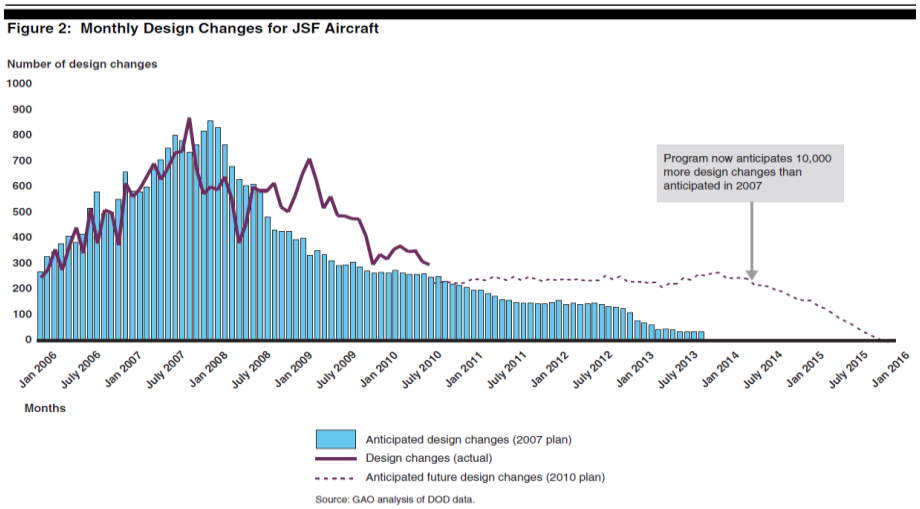


Figure 2. The rate of RV (%) and effort (hrs) throughout the *Project Release_B* lifecycle



Use Models to Improve Early Decisions

- "Current development processes allow 70% of faults to be introduced early in the life cycle, while 80% of them are not caught until integration test or later with a repair cost of 16x or higher." (Feiler et al.)
  - **Use early phase models that enable defect detection analyses.**
- "Focus on the interfaces between the software and the system in analyzing the problem domain, since these interfaces are a major source of safety-related software errors." (Lutz)
  - **Analyze sets of models to detect gaps and inconsistencies.**
- "…by the time a product is ready for development, over 90 percent of the operating and support costs have been determined." (GAO)
  - **Model system trade spaces to enable exploration and robust MDAO.**

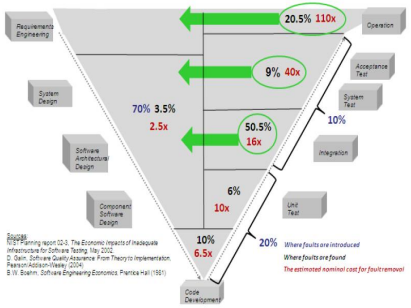
Adventium™ LABS

Make Better Early Decisions

"Current development processes allow 70% of faults to be introduced early in the life cycle, while 80% of them are not caught until integration test or later with a repair cost of 16x or higher."

(Feiler et al.)

Use early modeling with good error detection analyses.



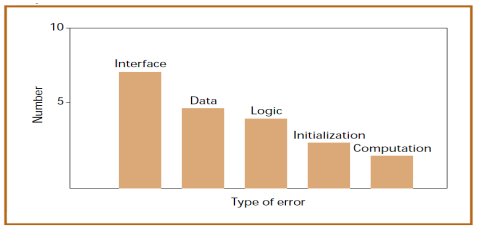


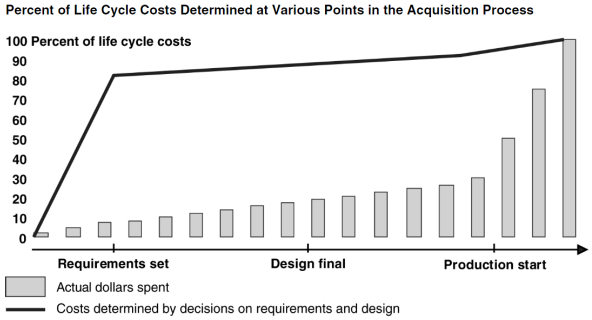
Figure 2. This Pareto chart example shows that Interface defects comprised the largest defect class.

"Focus on the interfaces between the software and the system in analyzing the problem domain, since these interfaces are a major source of safety-related software errors." (Lutz)

Detect gaps and inconsistencies between models.

"…by the time a product is ready for development, over 90 percent of the operating and support costs have been determined." (GAO)

Model system trade space for exploration and MDAO.



Adventium™ LABS

How Does FUSED Work? (A Peek Under the Hood)

- Extensible language technology
- Extended model project make
- Framework type/ontology system
- FUSED language execution

Silver and Extensible Languages

UMN Silver higher-order attribute grammar system

- Strongly typed functional programming features.
- Attributes may have as values attributed parse trees.
- Supports multi-phase translations.
- Modular grammars, e.g. mix-and-match extensions.
- Concise implementation of extensions via forwarding.

Used in FUSED to:

- Specify and implement extensions to existing languages.
- Specify framework type/ontology hierarchy.
- Generate pre-processor and post-processor tools.

Adventium™ LABS

UNIVERSITY OF MINNESOTA Driven to Discover℠

Example Boolean Expression Abstract Grammar

```
nonterminal Expr with eval, negation;
synthesized attribute eval :: Boolean;
synthesized attribute negation :: Expr;

production and
e::Expr ::= l::Expr r::Expr
{ e.eval = l.eval && r.eval;
  e.negation = or(not(l),not(r));
}
production or
e::Expr ::= l::Expr r::Expr
{ e.eval = l.eval || r.eval;
  e.negation = and(not(l),not(r));
}
production not
e::Expr ::= s::Expr
{ e.eval = !s.eval;
  e.negation = s;
}
```

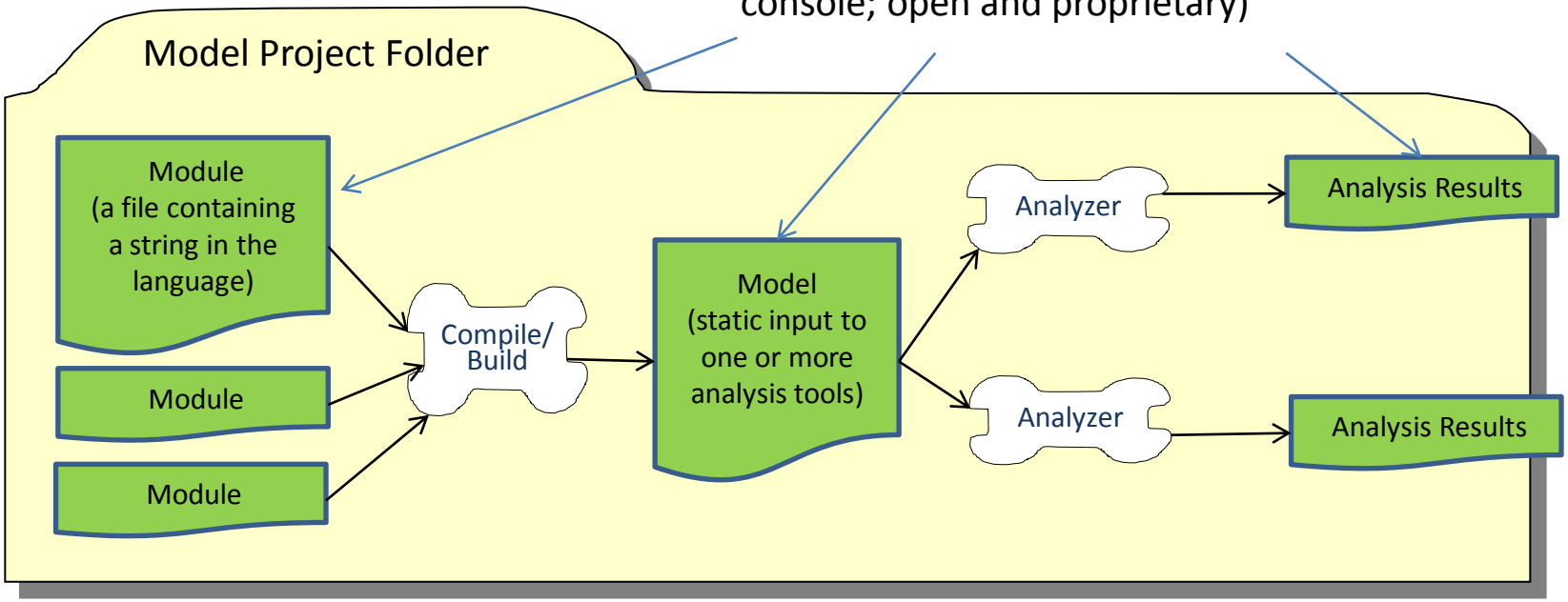
```
production literal
e::Expr ::= b::Boolean
{ e.eval = b;
  e.negation = literal(!b);
}
production implies
e::Expr ::= l::Expr r::Expr
{ forwards to or(not(l),r);
}
production iff
e::Expr ::= l::Expr r::Expr
{ e.eval = l.eval == r.eval;
  forwards to and(implies(l,r),
                  implies(r,l));
}
```

From “Integrating Attribute Grammar and Functional Programming Language Features,” Eric Van Wyk, International Conference on Software Language Engineering, 2011.

Adventium™ LABS

Models, Languages, and Environments

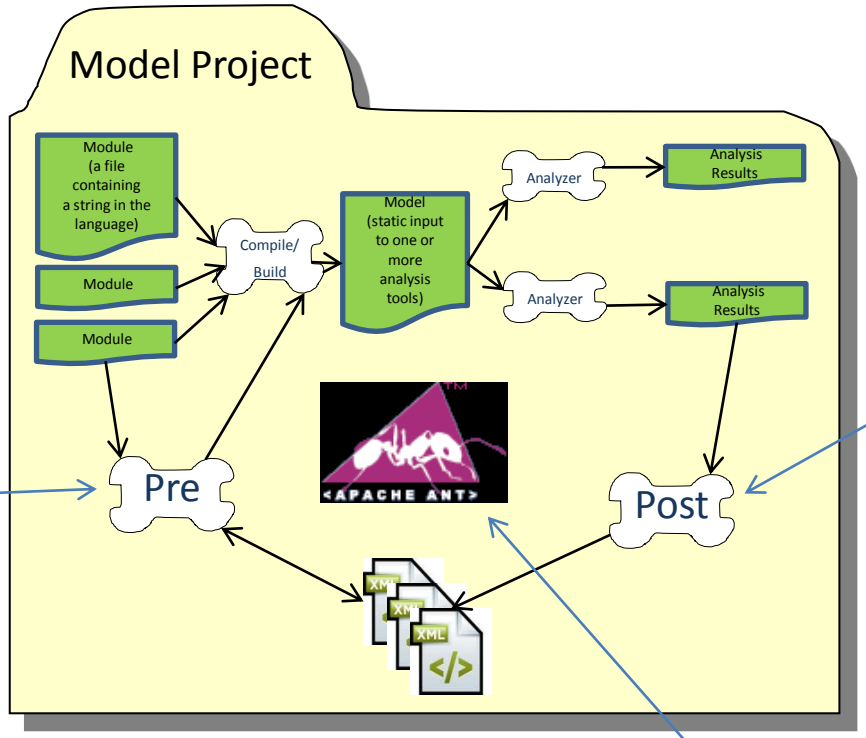
Typically distinct languages (messy in practice because a mixture of writable and read-only; text and GUI and console; open and proprietary)



The semantics within a selected modeling environment will be deeper within its specialized domain than that of any other language, including FUSED. We rely on these semantics and supporting tools to do the job within this domain.

When FUSED is extended to support a selected modeling environment, some of the semantics of that environment will apply to the FUSED extensions.

FUSED Additions to Model Projects



Model preprocessors (generated from Silver) support design choices and language extensions.

Model postprocessors (generated from Silver) publish abstractions of models and analyses.

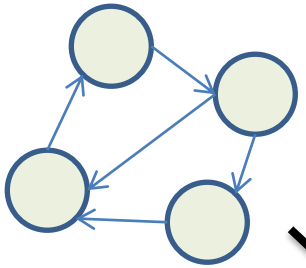
Parameterized ANT build script (callable from compiled FUSED specifications) orchestrate added capabilities.

- Manage FUSED data
- Invoke pre and post processors
- Invoke standard tools
- Manage dependencies & caches

Adventium™ LABS

FUSED Abstract Type Ontology/Hierarchy

FUSED Framework is extended with a set common abstract types and representations for elements that appear in multiple modeling languages and semantics.



Published model elements are abstracted to a common type and representation.

Elements are converted to the subscribing language with type-checking (native and extended).

AADL

SMTLib

Adventium™ LABS

Example FUSED Model Element Types

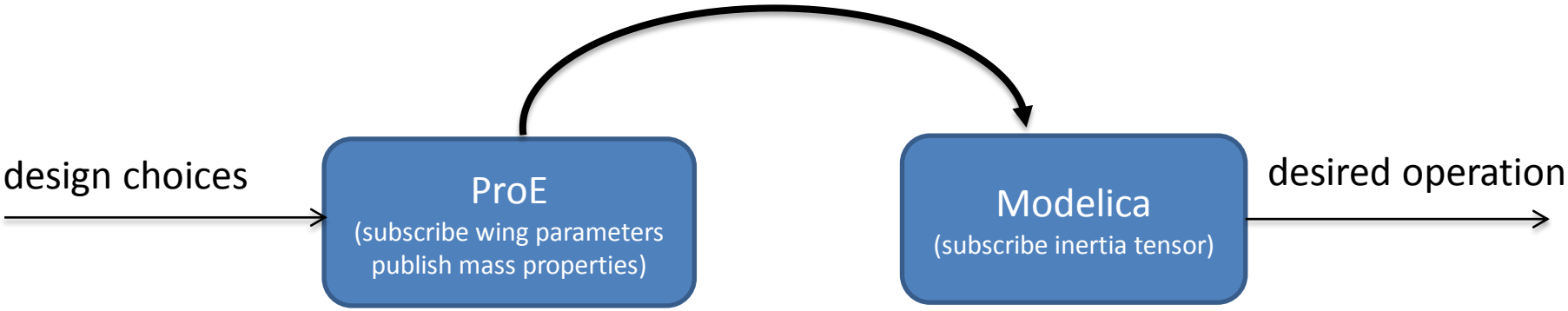
- Constructive types, e.g.
  - Float, Int, String
  - Array<T>
  - Constrained variables
- Type qualifiers (vaguely analogous to Java Interfaces), e.g.
  - Dimensions and units
  - Frame of reference
  - Interval uncertainty
  - Stochastic uncertainty
- Types of overall model abstractions, e.g.
  - Model structure graph of typed nodes and edges
  - Black-box function mapping design choices to quality metrics
  - timed data traces
- Users can combine and extend types as needed, e.g. to define discipline-specific ontologies if desired.

Adventium™ LABS

Notional FUSED Specification Execution

Pub/sub satisfaction rules (names, *types*)

- Pub/sub allows flexible model mix-and-match
- Not necessarily individual element-to-element
- Robust to model edits
- Small spec for automated large traceability



1. Determine design configurations
2. Determine needed subscriptions
3. Determine & invoke any needed publisher operations
4. Satisfy subscriptions (with type-checking)
5. Invoke desired operation

Adventium™ LABS

FUSED Models Across Multiple Sites