FUSED Framework for System Engineering
Hands-on Tutorial
SAE AADL 19 April 2012

Steve.Vestal@Adventiumlabs.com
Tutorial Agenda

• Overview: Goals and Approach

• Scenario-Driven Capabilities
  – Overviews
  – Hands-On Walk-Throughs

• Summary & Future Directions
Goal: A Tool and Process Framework for cyber-PHYSICAL System Engineering

Online additive manufacturing by Stratasys Redeye on Demand in Eden Prairie, MN
(Also see Proto Labs in Maple Plain, MN)
Multiple engineering disciplines address different aspects of the problem.

Each uses differing social, scientific, and mathematical methods, notations, and tools (often based on centuries of experience and development).

There is enormous investment in training, models, languages, tools, etc. (past and future).
Goal: Coordinate Layers of Abstraction

All specifications and models are abstractions used for specific purposes.

Is an abstraction good enough for its purpose?

What abstraction relations exist between different models?
Engineering processes are evolving to be more iterative, concurrent, life cycle oriented.

Requirements engineering is just one of the many disciplines involved.

Field operations and maintenance is just one of the many disciplines involved.
Approach: Select, Extend, Integrate Multiple Discipline-Specific Environments

Leverage existing specialized languages, tools, skills.
- Research, development, knowledge
- Training and familiarity
- Proven and accepted
- Tool development, past & future
- Existing validated models

FUSED meta-language provides novel capabilities for system engineers.
- Compose multiple specialized models
- Verify consistency across diverse models
- Automate system design & integration tasks
Control & Data Flows Between Models

FUSED Framework (with abstract types)

Penn State TSV

TOPCASED SysML

MS Excel

PTC ProE

MIT AVL

OSATE AADL

OpenModelica

ECliPSe MiniZinc

Z3 SMTLib

FUSED

Copyright © 2012 Adventium Labs
# Languages and Tools in Tutorial Download

<table>
<thead>
<tr>
<th>Language/Tool</th>
<th>Source</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>SysML/TOPCASED</td>
<td>TOPCASED consortium</td>
<td>open</td>
</tr>
<tr>
<td>AADL/TOPCASED+OSATE</td>
<td>TOPCASED, SEI</td>
<td>open</td>
</tr>
<tr>
<td>Trade Space Visualizer</td>
<td>Penn State</td>
<td>ROTS</td>
</tr>
<tr>
<td>Excel</td>
<td>Microsoft</td>
<td>COTS</td>
</tr>
<tr>
<td>Creo (ProE)</td>
<td>PTC</td>
<td>COTS</td>
</tr>
<tr>
<td>Athena Vortex Lattice</td>
<td>MIT</td>
<td>open</td>
</tr>
<tr>
<td>OpenModelica</td>
<td>Modelica consortium</td>
<td>open</td>
</tr>
<tr>
<td>SMTLib/Z3</td>
<td>SMT consortium/Microsoft</td>
<td>open/ROTS</td>
</tr>
<tr>
<td>MiniZinc/minizinc+ECLiPSe</td>
<td>U Melbourne/ECLiPSe org</td>
<td>open</td>
</tr>
</tbody>
</table>

FUSED framework is extensible. Level of support varies from tool to tool.

Copyright © 2012 Adventium Labs
What do Users/Developers Do?

Subject matter experts use the languages and tools they choose to do specialized engineering work.
- FUSED adds some extensions (e.g. pub/sub, type qualifiers)

System engineers use the FUSED language to specify and enact relationships between models, e.g.
- select sets of models to be composed
- configure model design choices
- publish/subscribe satisfaction rules
- invoke model-specific analyses/operations
- invoke design meta-tools (e.g. verifiers, optimizers)
- manage change/uncertainty/sensitivity propagations
How Does it Work?

• 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
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,”
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.

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

Model postprocessors (generated from Silver) publish abstractions of models and analyses.
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
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
- Function mapping design choices to quality metrics
Notional FUSED Specification Execution

Pub/sub satisfaction rules (names, \textit{types})
\begin{itemize}
\item Pub/sub allows flexible model mix-and-match
\item Not necessarily individual element-to-element
\item Robust to model edits
\item Small spec for automated large traceability
\end{itemize}

\begin{enumerate}
\item Determine design configurations
\item Determine needed subscriptions
\item Determine & invoke any needed publisher operations
\item Satisfy subscriptions (with type-checking)
\item Invoke desired operation
\end{enumerate}
Tutorial Agenda

• Overview: Goals and Approach

• Scenario-Driven Capabilities
  – Overviews
  – Hands-On Walk-Throughs

• Summary & Future Directions
Basis of Tutorial Scenario: Tactical Surveillance UAV

UAV used by ground troops for local surveillance.

Choice of payloads

Backpacked in pieces, assembled at site

Hand-launched

Lockheed-Martin Desert Hawk
Models and Languages in Scenario

- Requirements (SysML)
- Equations Model (Excel)
- Design Optimization (MiniZinc)
- Mixed Fidelity Model (Excel)
- Trade Space (ATSV)
- Solid (Creo/ProE)
- Aerodynamics (AVL)
- Vehicle Dynamics (Modelica)
- Model Consistency (SMTLib)
- Avionics (AADL)

12 domain-specific models (with multiple configurations)
9 domain-specific languages (plus FUSED META-Language)
5 FUSED composition specifications (plus subspecs)

Copyright © 2012 Adventium Labs
Walk-Throughs

1. Trade space exploration of requirements + equational model
2. Trade space exploration of requirements + equational + solid geometry + aerodynamics models
3. Simulation of solid geometry + aerodynamics + dynamical systems models
4. Mixed-initiative trace space exploration + design optimization models
5. Verification of consistency property between solid geometry + avionics architecture models
Hands-On: Explore Trades with Requirements and Abstract System Models
Hands-On: Dynamical Simulation

Wing Option #1: 4412 (long wing)
Wing Option #1: 2412 (short wing)

Solid Model (Creo/ProE) → Aero (AVL) → Stability Derivatives → Mass Properties, Center of Gravity, Moments of Inertia, etc. → Vehicle Dynamics (Modelica)

Copyright © 2012 Adventium Labs
Hands-On: Explore Trades with Multi-Level Models

- Requirements (SysML)
- Solid Model (Creo/ProE)
- Aero Model (AVL)

Wing Option #1: 4412 (long wing)
Wing Option #1: 2412 (short wing)

- Hands-On: Explore Trades with Multi-Level Models
- Mixed Fidelity (Excel)
- Detailed Models
- Abstract Models

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

Engineer uses MiniZinc/ECLiPSe to automatically optimize the design for the customer’s higher-level trade-off choices.
Hands-On: Model Consistency Verification

Verify all avionics objects and relations of specified types are subgraph isomorphic to the solid model, subject to type compatibility constraints.
Consistency Verification Benchmarks

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

- Convention in SMT community seems to be to use scripts to generate basic (human un-readable) inputs. Using Silver to easily create domain-specific extensions of standard SMLib may be a big help.
Tutorial Agenda

• Overview: Goals and Approach

• Scenario-Driven Capabilities
  – Overviews
  – Hands-On Walk-Throughs

• Summary & Future Directions
Future Directions

• FUSED language & tool (e.g. composition & navigation features)
• Abstraction and typing capabilities, semantics, structures
• Framework (e.g. systematic plug-in develop & install)
• Change, uncertainty, sensitivity propagation
• Site integration (e.g. DCCM, PLM, SOA model servers)
• Standard types and ontologies
• Library of domain-specific environment plug-ins