FUSED: A Tool Integration Framework for Collaborative System Engineering

Mark Boddy, Martin Michalowski, August Schwerdfeger, Hazel Shackleton, and Steve Vestal
Adventium Enterprises
Minneapolis, MN, USA
{*mark.boddy,martin.michalowski,august.schwerdfeger,hazel.shackleton,steve.vestal*}*@adventiumenterprises.com*

***Abstract*—FUSED is a tool integration framework that supports multiple engineers who are collaborating in the development of a diverse set of engineering models used for multiple purposes in multiple phases of development. FUSED is extensible to support a chosen set of modeling environments; a few examples from our work are requirements, solid geometry, computational fluid dynamics, dynamical systems, and vetronics/avionics. An extensible language approach is used, so that many FUSED capabilities are presented to domain experts as minor additions to familiar languages and tools. There is also a special FUSED language to specify compositions of models. Compositions may be used for multiple purposes, *e.g.*, to specify multiple views of a component, verify inter-model consistency, specify part/whole assemblies, or apply design operations to models. One goal of FUSED is to reduce errors due to inconsistencies and emergent properties that occur across multiple models being developed by multiple domain-specific experts. For example, FUSED has an extensible typing and meta-typing system, and compositions may include powerful model verification environments. Another goal is improved support for concurrent, collaborative, mixed-initiative, evolutionary development processes. For example, FUSED was designed to support dependency tracking, change management and ripple effects analyses, version control and remote model server access, and mixed-initiative and multi-disciplinary collaborative optimization.**

***Keywords*-tool integration; system engineering; collaborative development; extensible languages;**

I. Introduction

We use the term "modeling environment" to mean a set of languages used to model systems from particular viewpoints plus a set of tools that automate associated engineering tasks. Some examples (all freely available) are OpenModelica for dynamical systems, BRL-CAD or Google SketchUp for solid modeling, Athena Vortex Lattice for computational fluid dynamics, TOPCASED for SysML requirements and AADL vetronics/avionics models. A modeling environment typically has a primary language for human entry and a number of additional data representations. The primary language could be a standardized textual language specified by a formal grammar, but it may also be specified informally by a GUI or API. Environments also include other representations, such as those generated by various model analysis and simulation tools. Our tool integration framework, which we call FUSED, has been designed so that it can be extended to support any given set of modeling environments.

The FUSED framework provides additional capabilities to specify compositions of diverse models, to script operations on these compositions, and to support dependency tracking and change management in a collaborative development environment. Examples include a composition of solid and dynamical models to capture different viewpoints of a component; a composition of components into an assembly or architecture; or a composition of a design optimizer with a design.

Developers must be able to specify how these new capabilities are to be applied. Our approach is to leverage existing modeling environments as much as possible by applying extensible language concepts. New FUSED capabilities are provided to domain experts as minor extensions to familiar languages. There is an additional, new FUSED language for specifying compositions of models developed in other modeling environments. This language deals with issues that are inherently multi-model and multi-environment and is primarily intended for system rather than domain-specific engineering.

Our concept of model includes traditional design models such as geometry models of aircraft wings and DAE models of vehicle dynamics. It also encompasses what might be called design process models, which are models that operate on other models. Examples in this category are design optimizers and trade space visualizers, model checkers and verifiers, and uncertainty propagators and global sensitivity analyzers. FUSED compositions can assemble parts into assemblies; *e.g.*, four wheels and a framework into a chassis. They can also specify applications of process models to design models; *e.g.*, a composition of a design optimizer or trade space visualizer with another composition of design models.

Defining a semantics for a composition of models that themselves have diverse and incommensurate semantics is a

This work was supported by DARPA under the META program contract # FA8650-10-C-7076. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Approved for Public Release, Distribution Unlimited. Submitted 2^{nd} Workshop on Analytic Virtual Integration of Cyber-Physical Systems.

challenge. For example, the extensive and intricate geometric semantics defined for a solid modeling environment are at best weakly reflected in logical languages found in vetronics and avionics environments. Our approach is to combine concepts found in classical type theory with concepts found in theories of abstraction. The semantics of FUSED compositions are the abstractions of types found in the different models, at a level of abstraction that is common across the models.

We assume FUSED will be used in a development process involving many developers who are concurrently modifying models from many modeling environments. These models may be used at different phases and exist at multiple levels of abstraction, *e.g.*, preliminary requirements through as-built verification and validation (V&V). We assume models may be stored in a mixture of version control systems or accessed using a remote model server.

To ensure its utility in this kind of development, the FUSED framework is extensible to support any selected set of modeling environments and any selected set of object types from those environments. For the design and implementation of the FUSED framework itself, there are a number of technologies from which to choose as a basis. The UML meta-language approach and its associated representations and tools provide an obvious example [8]. Semantic web (web 3.0) concepts and representations are another option. FUSED supports UML domain languages like SysML and uses XML for tool data exchange, and the framework is designed to support web/SOA-style interactions such as compositions involving remote model servers, but our primary implementation technologies are higher-order attribute grammars and extensible language methods; specifically, the Silver attribute grammar specification language and code generation tools [2]. Silver, although based around the paradigm of a parser generator, has the full power of a functional programming language; attributes on syntax-tree nodes in Silver may have values that are themselves full syntax trees, and there are also formalisms and analyses for the concise specification and seamless composition of sets of language extensions, both on the syntactic and semantic levels.

The following sections provide overviews and illustrative usage scenarios for problems addressed, model composition semantics, modeling environments and compositions, concurrent collaborative development, and the architecture and implementation approach for the FUSED framework itself.

II. Problems Addressed

Many errors occur due to inconsistencies between models developed in different environments, and to misunderstandings by engineers trained in different disciplines and focusing on different issues. Because all models are abstractions, they represent only aspects of the system, developed to deal with particular sets of issues for particular purposes. There are characteristics of a system (sometimes called emergent properties) that are really captured only in sets of models. We want to reduce errors that occur due to inconsistencies between models or due to failure to consider interactions between models. Section III discusses some FUSED features that address these problems and presents an illustrative usage scenario.

Development is a mixed-initiative process. Automated design tools are becoming increasingly common and powerful, but these must ultimately be applied by human developers and as such the human/machine interactions are becoming increasingly complex. FUSED makes it easy for developers to apply automated design aids to complex models to support complex, mixed-initiative development activities.

Additionally, development processes are becoming increasingly collaborative, concurrent, and distributed. The waterfall model is being supplanted by processes in which product lines undergo continuous evolution throughout their life cycle. Advanced multi-disciplinary system engineering frameworks need to be well-integrated with supporting processes like distributed revision control, collaborative model development, and change management. FUSED integrates with advanced process support tooling to create an overall environment in which developers can stand on each other's shoulders instead of each other's toes. Section V discusses some FUSED features that address these issues and presents an illustrative usage scenario.

III. Composition Semantics

Our definition of "semantics" is "a way to map a string of symbols to a structure defined in some field of mathematics or science." Examples would include mapping a STEP file to a structure in solid geometry or mapping a string in the Modelica language to a system of hybrid differential algebraic equations.

For a particular modeling environment, we have to live with what we are given. This means we have to deal with different modeling environment semantics mapping strings to different mathematical domains, using different methods to define these mappings with differing degrees of rigor, *etc.* For example, a solid modeling semantics maps models to solid geometric objects with semantic rules such as "no two distinct solid parts may overlap in three space." These concepts are absent in the Modelica language definition. However, it is possible to map a moment-of-inertia tensor from the solid geometry domain into the domain of differential algebraic equations. There are some semantics that are explicitly defined only in the solid modeling environment, such as the concept of inertia of a material object (which is arguably a scientific rather than a purely mathematical semantics). But there are some semantics common to both modeling environments, such as units and linear operations on vectors in three space.

When two models are composed in FUSED, a developer can specify that one model may *publish* (pub) an object that is used to satisfy a *subscribe* (sub) in another model. FUSED automates this process and uses strong type-checking as part of what might otherwise be a manual cut-and-paste operation. When an object is specified for publication, what is actually published is a string of symbols whose semantics are an abstraction that is common to both modeling environments. In the case of a moment-of-inertia tensor, this is a 3×3 matrix of floating point numbers with associated units. Using our concept of language extension, FUSED provides additional information to this pub/sub process. For example, we extend the Modelica language slightly to include the concept of a frame of reference, in which case FUSED provides more semantics and type checking than standard Modelica.

In any particular FUSED installation, a set of common abstract types are defined. This means there are abstraction relations from types of objects in the various supported modeling environments to and from these common abstract types, made concrete in tools that can extract objects from various modeling languages and convert them to and from a common representation. In principle, the semantics of a pub/sub relation in a FUSED specification involves a mapping to a mathematical domain common to the two modeling environments, plus an abstraction relation for each of the two modeling environments based in theories of abstraction that would typically be different for the two modeling environments. Arguments then need to be made that operations on a subscribing model are correct; *i.e.*, information lost in the abstraction process does not render model analysis results invalid. In current practice, we just write Silver specifications for the common representations and conversion tools. This is an area rife with opportunity for further theoretical research and improved rigor.

Figure 1 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, the system engineer would like some assurance that the solid model of an aircraft is consistent with the avionics model. In particular, the engineer would like assurance that the logical hardware resources and connections in the avionics model are consistent with electronics boxes and cables in the solid model.

Using the FUSED composition language, the system engineer specifies that an abstract representation of the static structure of a model be published for both models. This is a graph in which nodes and edges may have one or more types specified, where the set of node and edge types is just a list of identifiers specified in each model. (It is typical to provide language extensions that allow model developers to specify additional typing information if the standard language mechanisms are not sufficient.) For example, in the published abstraction of a solid model, nodes represent parts and assemblies and edges represent containment and attachment relations.

What the system engineer wants is assurance that there is a subgraph isomorphism between the logical hardware elements and connections in the avionics model and parts in the solid model, where the types of corresponding nodes and edges satisfy a specified type compatibility relation (*e.g.*, logical processors and buses in the avionics model map to LRU and cable parts in the solid model). This property can be concisely specified in SMTLib, a standard language agreed upon by the Satisfiability Modulo Theories research community for which there are a number of tools. This specification subscribes to two abstract model structures, and the SMTLib modeling environment provides a "check satisfiability" operation. By periodically running this FUSED composition specification, the system engineer receives ongoing assurance that this consistency property is maintained between the two models as they evolve during development.

IV. Model Environments and Compositions

A model environment is a set of languages and a set of tools that support a particular engineering discipline. An example is OpenModelica [6], to which the primary human input language is standard Modelica, and whose toolset includes a compiler and a simulator/solver. Any representation that holds data of interest at the system engineering level is also a language that FUSED at least needs to be able to read. For example, simulation trajectories and linearizations contain data of interest.

A specific model or related set of models is stored in a folder or project that is associated with a particular modeling environment. For example, the concepts of a modeling environment and an Eclipse project type are effectively synonymous, and there is an Eclipse plugin to support the OpenModelica project type.

Extending FUSED to support a particular modeling environment involves the following activities.

- Identify types of model elements that should be made visible at the system engineering level (in FUSED composition specifications). Ideally, many of these will already exist in the FUSED common type system; otherwise, they must be added.
- Define extensions to the primary language to support FUSED capabilities. Two near-universal extensions are the ability to declare that elements of a model are to be published (*i.e.*, made visible for use in FUSED composition specifications) or subscribed (*i.e.*, provided to a model when it is used in the context of a particular FUSED composition specification). Other common extensions allow additional typing information to be declared or support parametric/configurable models.
- Identify operations that can be performed on the model using the available modeling environment tools. (Mixed initiative operations that require some interaction with

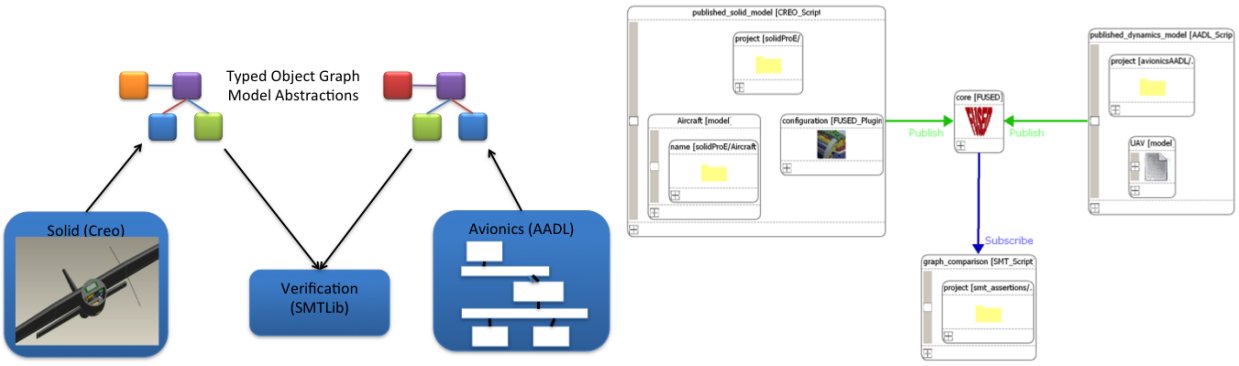

Figure 1. Model consistency usage scenario and FUSED composition specification

a domain expert should be permitted, although we have not demonstrated that to date.)

- Develop wrapper specifications in the FUSED language to present a set of convenient interfaces to higher-level FUSED programmers. This is not essential, but often makes the model more easily used by system engineers who are not familiar with either the model structure or the modeling environment.

The final section of this paper outlines how we implement these extensions. Basically, we do this by extending the existing build procedure to perform the identified operations automatically. Invocations of existing tools are wrapped with invocations of preprocessors and postprocessors that implement new FUSED capabilities and provide a proper interface into the FUSED framework.

When models are composed in a FUSED specification, that specification declares a pattern of publish and subscribe relationships between models together with other information such as scripting of operations to be performed. It would be quite tedious to declare this for every individual model element, and declaring a dependency for every individual element would also make composition specifications less robust to model edits. FUSED publishes structured sets of model elements, and FUSED subscribes are satisfied using a name space search procedure. The FUSED composition language includes a set of operations to perform restructurings, renamings, *etc.* (in the figures that show FUSED composition specifications, that is what the boxes labeled "FUSED" represent).

There is always a risk of mismatch, but this risk exists even if dependencies were manually specified for every individual scalar (which adds risk due to losing information about structural relationships between elements). FUSED mitigates this risk by providing a fairly powerful type system. In addition to basic types and type constructors like floats, integers, and arrays, FUSED supports the concept of a type qualifier. A type qualifier is additional information such as units, frames of reference, or uncertainties, that can also be associated with a model element. Modeling language extensions can be used to increase the amount of typing information provided and checked, and type qualifier operations are available in the FUSED composition specification language.

Figure 2 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, three domain-specific models are developed to represent a UAV from three viewpoints. A solid model is used to capture and analyze the physical structure, a computational fluid dynamics model is used to analyze aerodynamic forces on the vehicle, and a dynamical systems model is used for flight dynamics. The models are configurable, so that system engineers can explore various combinations of design parameter choices.

The vehicle dynamics model needs data that can be obtained from a mass properties analysis of the physical structure, such as mass, moment-of-inertia tensor, and control surface areas. The vehicle dynamics model also needs a set of stability derivatives computed at a number of trim points, which can be obtained by CFD analysis. Rather than cut-and-paste for each UAV configuration, the vehicle dynamics model is modified slightly so that it subscribes to these values, which can be published by the other models. When the FUSED composition specification is executed, the models are configured, analyses executed, and data converted and copied as needed to obtain the specified vehicle dynamics analysis results; *e.g.*, a set of simulation traces. (This is similar in principle to what can be done with some existing commercial tools. A difference in this example is that FUSED supports complex object types with strong type checking.)

V. Development Processes

Models and associated assets are traditionally managed using revision control systems such as Subversion or Git. The design of FUSED also supports remote model servers. In order to execute a FUSED composition script, all that is needed is the ability to send a block of data to satisfy model subscriptions and receive a block of data published

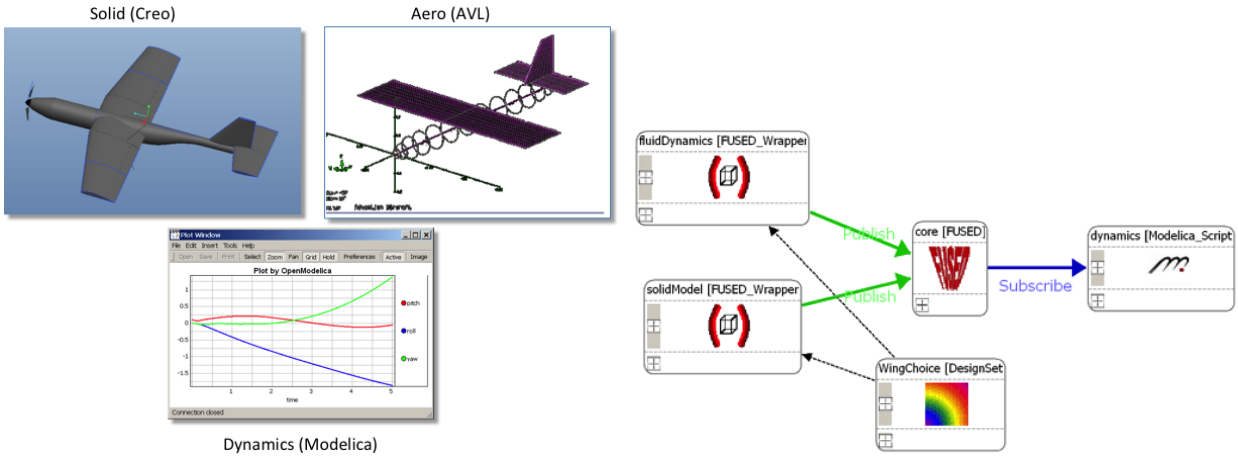

Figure 2. Model configuration and pub/sub scenario with FUSED composition specification

by a selected operation. To make full use of FUSED (*e.g.*, modeling language extensions), the model servers should also host a FUSED framework.

The set of all FUSED composition specifications in a development project captures much semantically rich information about the relationships and dependencies among all the models. Moreover, we conjecture that this information is captured much more concisely and manageably — the number of keystrokes needed to enter publish and subscribe declarations in models and abstract pub/sub dependencies in composition specifications is relatively small compared to the list of detailed element-to-element dependencies enumerated by the FUSED tools as it processes composition specifications. Moreover, abstract pub/sub relationships are robust with respect to changes in the individual models and thus more easily maintained. Finally, the fact that FUSED composition specifications are subjected to a variety of automated analyses provides some assurance of the correctness of these relationships.

The dependency relationships between models that are captured in FUSED composition specifications can be used in a variety of ways. Firstly, FUSED builds on traditional build/make dependency tracking technology. As FUSED composition specifications are executed, operations on individual models are only invoked as needed to provide up-to-date published data. Secondly, FUSED encourages and supports the use of configurable models. When an operation is performed on a model, this means the results depend on the values used to satisfy subscriptions in that model. We have prototyped results caching, so that repeats of an operation need not be done if the results of an earlier matching analysis have been cached.

Models are created for parts of the system in order to understand their properties from a particular viewpoint (*e.g.*, a Modelica model) to understand certain kinds of dynamical behaviors. We also want to support models that can be used as part of the design process itself, models that can be applied to other models. Examples of this are trade space models to support trade space exploration and Pareto frontier identification, various kinds of design optimization methods, and model and multi-model verification and validation activities. In cases like this, a FUSED composition is not an assembly of parts that is analyzed to determine properties of the assembly; it is executed to carry out a design activity on a model or composition of models. To support this, FUSED can publish and subscribe types of elements that are abstractions of models themselves; *e.g.*, the abovementioned typed object graph, as well as constraints and properties.

A variety of process models have been developed to perform uncertainty propagation and global sensitivity analysis to understand how model evaluation metrics depend on model design configuration parameters [3]. We are particularly interested in combining this sort of analysis with our dependency tracking capabilities to perform smart ripple effects analysis — is a change in a model big enough to significantly impact other models that relate to it in some way?

Figure 3 depicts a usage scenario and FUSED model composition specification that illustrate some of these concepts. In this scenario, the requirements engineering team is wrestling with the trade-offs between quality metrics like range, endurance, payload capacity, and cost. They are also still uncertain about what the available technology will make possible. What they would like to do is use a tool like Trade Space Visualizer [4] to explore the trade space and identify the Pareto frontier for the range of possibility given current technology.

The aeronautical engineering team has created an initial equational model (a spreadsheet) that allows these metrics to be estimated for a variety of design alternatives; *e.g.*, different choices of wing structure, batteries, motors, and propellers. If the aeronautical engineers knew the final re-

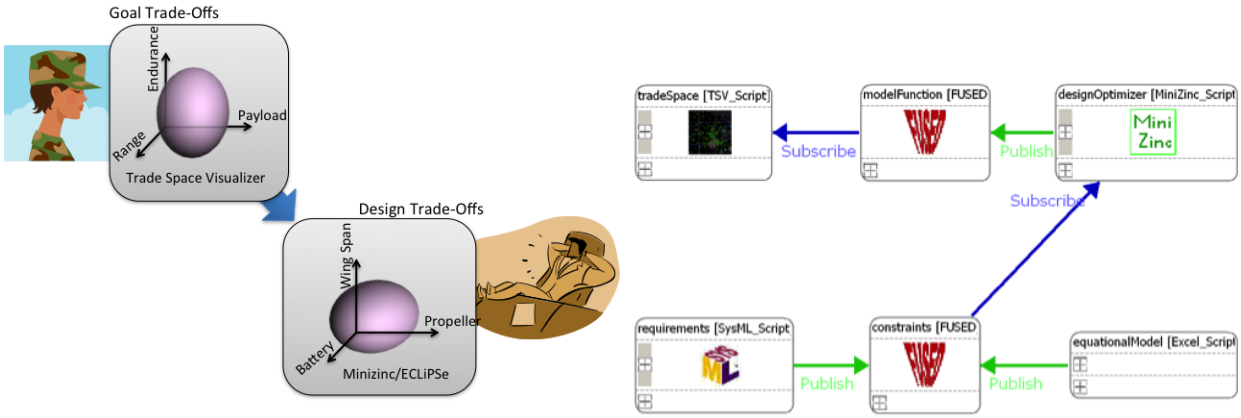

Figure 3. Mixed initiative design optimization scenario and FUSED composition specification

quirements for range, endurance, *etc.*, then they could select design choices that optimize the vehicle for the selected requirements. In fact, the aeronautical engineers constructed an optimization model (written in the widely-used MiniZinc language [5], for which a number of tools are available) that automatically makes good choices when given a final set of requirements. Such design decisions are irrelevant to the work of the requirements engineering team, who are merely interested in what could be achieved in terms of cost and endurance if they decided that low cost and high endurance are the dominant concerns of the end users.

To support these activities, the system engineering team creates a FUSED specification that composes the MiniZinc design optimization model with the requirements and equational models to create what is essentially a self-optimizing aircraft model that is parameterized by range, endurance, payload and cost — the parameters the requirements engineering team wants to explore. Trade Space Visualizer provides a variety of sampling methods that can be applied to an abstraction of a design model in which that model is a function that can be evaluated for a set of input parameters to determine the values of a set of evaluation metrics. In this FUSED composition specification, this is an abstraction that can be published by a model. The Trade Space Visualizer model subscribes to the functional abstraction that is published by the design optimization model. When this FUSED specification is executed, the Trade Space Visualizer will perform a statistical sampling of the design space by invoking the design optimization model/function, identify the points on the Pareto frontier, and provide a variety of graphical display formats that allow the requirements engineering team to explore this space.

VI. Extensible Framework

In this section, we briefly overview the current FUSED framework implementation architecture and technologies.

We currently use TOPCASED [7] for both its SysML and AADL modeling environments. TOPCASED is based on Eclipse, and one can think of an Eclipse project type as a modeling environment type. Figure 4 illustrates a set of model projects for a variety of modeling environments.

A central abstraction for a model is a set of operations that map a set of subscribed values to a set of published values. (This is an abstraction that can itself be published, *e.g.* for use by sampling-based uncertainty propagation models or gradient-search design optimization models.) These operations are implemented as a set of targets in a build script template that is created when FUSED is extended to support a particular modeling environment. The build scripts wrap calls to existing modeling tools with calls to preprocessors and postprocessors to handle language extensions, publish and subscribe operations, *etc.* Development of these preprocessors and postprocessors is another task performed when extending FUSED to support a particular modeling environment. There are currently cases where build scripts need to be tweaked manually to handle configurable models

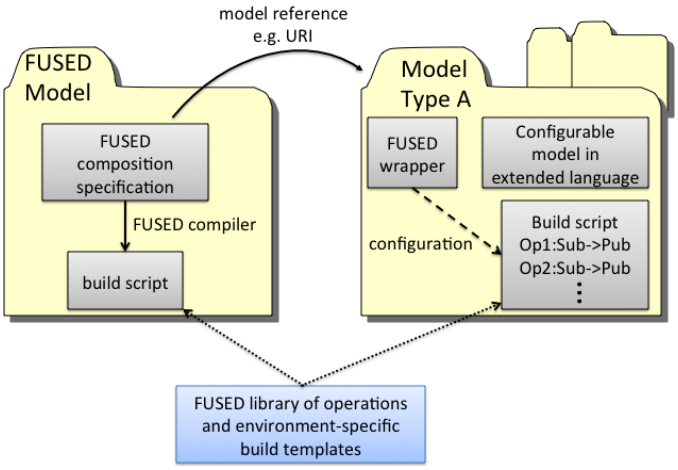

Figure 4. FUSED Project Types and Relationships

(acceptable to a software engineer but not to a system engineer), but eventually such configuration, when needed, should be generated automatically from a FUSED wrapper specification for a configurable model.

Specifications written in the FUSED composition language are models just like any other; *e.g.*, they can themselves be composed with other models. There is a modeling environment for the FUSED composition specification language itself. The build scripts here are not taken from a library, however; they are entirely generated from FUSED specifications by a FUSED compiler. When executed, these scripts mix calls to other model project builds with calls to various FUSED operations; *e.g.*, to perform restructurings on structured sets of published and subscribed model elements. Data is exchanged using an XML common type representation format, with conversions and type checking being performed by the preprocessors and postprocessors associated with each modeling environment.

The common types and their representations and conversion routines, and the preprocessors and postprocessors, are specified in the attribute grammar specification language, Silver. Attribute grammars are well-suited to concise specification of complex languages and data representation structures. Silver provides a rich typing system (*e.g.*, attributes on nodes of syntax trees may themselves be syntax trees, and generic attribute types are supported), so there is plenty of power to specify complex checks and conversions. Silver also has features supporting seamless language extension. This is chiefly done by specifying additional rules that recognize extensions to a language, and then placing "forwarding" constructs within those rules to specify a translation from the extended language back to the original. For example, a Silver-specified preprocessor would handle a subscribe declaration (an extension to the original language) by forwarding it to (rewriting it as) a declaration containing a literal value in the syntax of that language, after doing the appropriate look-ups and type-checks and conversions from a FUSED XML file. This re-writing is performed at the level of abstract syntax trees, rather than strings, allowing for more precise and fine-grained translations.

In summary, extending FUSED to support a specific modeling environment entails developing three kinds of new assets.

- Preprocessors and postprocessors for selected language representations, written in Silver. These implement language extensions such as publish and subscribe declarations.
- An ANT build template whose targets correspond to all the operations supported on models of that type. The template may contain subscriptions as needed to handle configurable models and parametric analyses.
- New common abstract types, including type-checking and conversion algorithms, may need to be specified in Silver and added to the FUSED common type system.

VII. Discussion

There is a trade-off involved in providing FUSED with a relatively powerful typing and language processing capability. This makes extending the framework more complex than if all it did was automate cut-and-paste of strings. On the other hand, it provides much more power. In our own demonstration exercises, we had unintended mismatches of units and frames of reference detected by our own framework. (Both of these classes of design defects are known to have resulted in a number of engineering failures and are still considered by many to be a source of significant risk.) More than this, though, is the potential for new capabilities like verification of non-trivial consistency conditions between models and support for more general mix-and-match of design process methods such as mixed-initiative multi-disciplinary design optimization. We conjecture that the potential benefits outweigh the additional extension complexity, which is a one-time overhead for each modeling environment in any case.

We believe that our choice of higher-order attribute grammar and language extension technologies was a good one, as they provide the added ability to compose models written in different languages by simply augmenting models slightly using simple syntax. We have not found it inconvenient to deal with UML languages via their XMI representations, and we have at least as much power to work at the meta-language and meta-type levels. Our language extension and pub/sub methods seem able to handle fairly complex model synchronization problems. The relationship to (semantic) web technologies is a bit more nuanced, and we currently tend to think of the relationship as one that invites synergistic integration. It is a combination that is needed to provide good support for distributed, concurrent, collaborative development. As one might expect, Silver is best suited for handling textual languages and representations having well-defined syntax, but relatively weaker for languages defined informally by the GUI or API of a specific tool.

Additionally, the overhead introduced by FUSED is minimal. While we have not conducted a thorough evaluation of the overall runtime impact, we observe that the time taken to translate FUSED-extended models to their equivalent base language is negligible and there is no impact on the simulation runtimes for these translated models when compared to the originals. Furthermore, while the process to extend FUSED to support a new modeling environment is in itself time consuming, it is a one-time effort. Assuming the software engineer develops a robust set of ANT build templates, their invocation will require no additional changes except for special cases. In these special circumstances, the software engineer only needs to modify the existing template rather than creating a new build script from scratch (a common process today).

FUSED is a work in progress. Our initial set of exten-

Table I
MODELING ENVIRONMENT EXTENSIONS

Category	Language	Toolset
requirements	SysML	TOPCASED
trade-off studies	(tool-specific)	Trade Space Visualizer
design optimization	MiniZinc	minizinc, ECLiPSe
spreadsheet	(tool-specific)	Excel
solid/geometric	(tool-specific)	Creo/ProE
fluid dynamics	(tool-specific)	Athena Vortex Lattice
dynamical systems	Modelica	OpenModelica
avionics/vetronics	AADL	TOPCASED/OSATE
model verification	SMTLib	Z3

sions supports nine domain-specific modeling environments (shown in Table I), but the set of language extensions and the set of common FUSED types is not what would be desired in a full system engineering project. The current tools for the FUSED model composition language only support publish and subscribe relations between models and combinations of operations provided by the associated environments, which supports only some of the various kinds of compositions discussed earlier. The FUSED implementation needs additional features — *e.g.*, occasional hand-configuration of build templates, acceptable to a software engineer but probably not to a system engineer — should be automated. As with all development projects at this phase, maturation is needed in the areas of documentation, error reporting, refactoring to facilitate easier extension, and general defect reduction.

Additionally, FUSED composition specifications capture much detailed information about dependencies between models, and we are developing ways that FUSED can use this data. We want to support specification and use of configurable models — component suppliers should provide a space of capabilities to system engineers who are making system-level trade-offs [1]. We want to support smart inter-model ripple effects analysis in which developers can analyze the magnitude and significance of changes or uncertainties in one model on other models. We want to support verification of consistency between models. We want to support highly parallel development processes, to the point that requirements engineering may be just another activity that goes on throughout a product and product line life cycle.

We have mentioned several areas where we believe that further research could yield significant benefits. Among these are an improved theory of semantics and abstraction, methods for smart change propagation analysis based on uncertainty and other "close enough" type qualifier information, experience with and improvement of inter-model consistency verification methods, improved caching of subscription-dependent model analyses, and more synergistic integration with advanced configuration management and model server and program management technologies.

VIII. ACKNOWLEDGMENTS

We would like to thank the Minnesota Extensible Language Tools group at the University of Minnesota for their Silver support, Lockheed-Martin for their aeronautical engineering and model development support, and the Defense Advanced Research Projects Agency (DARPA) for supporting this work under the META program contract # FA8650-10-C-7076.

REFERENCES

[1] Durward K. Sobek II, Allen C. Ward, and Jeffrey K. Liker, "Toyota's Principles of Set-Based Concurrent Engineering," *Sloan Management Review*, 40, 2, Winter 1999.

[2] Eric Van Wyk, Derek Bodin, Jimin Gao and Lijesh Krishnan, "Silver: an Extensible Attribute Grammar System," *Science of Computer Programming*, Elsevier Science, January 2010. See also http://www.melt.cs.umn.edu/.

[3] Christopher Hoyle, Irem Y. Tumer, Tolga Kurtoglu, and Wei Chen, "Multi-State Uncertainty Quantification for Verifying the Correctness of Complex System Designs," *Proceedings of the ASME 2011 International Design Engineering Technical Conferences & Computers and Information in Engineering Conference*, Washington, DC, August 2011.

[4] Stump, G.M., Lego, S., Yukish, M., Simpson, T. W., Donndelinger, J. A., "Visual Steering Commands for Trade Space Exploration : User-Guided Sampling With Example," *Journal of Computing and Information Science in Engineering*, 2009. See also http://www.atsv.psu.edu/.

[5] Peter J. Stuckey, Ralph Becket, Sebastian Brand, Mark Brown, Thibaut Feydy, Julien Fischer, Maria Garcia de la Banda, Kim Marriott, and Mark Wallace, "The Evolving World of MiniZinc," *MODREF 2009*. See also http://www.g12.cs.mu.oz.au/minizinc/.

[6] OpenModelica, the Open Source Modelica Consortium, http://www.openmodelica.org/.

[7] TOPCASED, the TOPCASED consortium, http://www.topcased.org/.

[8] K. Czarnecki and S. Helsen, "Feature-based survey of model transformation approaches," *IBM Systems Journal*, Vol 45, No 5, 2006.