

# Simulation-Based Design Using SysML

## Part 1: A Parametrics Primer

Russell S. Peak<sup>1,†</sup>, Roger M. Burkhart<sup>2</sup>, Sanford A. Friedenthal<sup>3</sup>, Miyako W. Wilson<sup>1</sup>, Manas Bajaj<sup>1</sup>, Injoong Kim<sup>1</sup>

<sup>1</sup> Georgia Institute of Technology  
<http://www.pslm.gatech.edu/>  
<http://eislabs.gatech.edu/>

<sup>2</sup> Deere & Company  
<http://www.johndeere.com/>

<sup>3</sup> Lockheed Martin Corp.  
<http://www.lockheedmartin.com/>

Copyright © 2007 by Georgia Tech, Deere & Co., and Lockheed Martin Corp. Published and used by INCOSE with permission.

**Abstract.** OMG SysML™ is a modeling language for specifying, analyzing, designing, and verifying complex systems. It is a general-purpose graphical modeling language with computer-sensible semantics. This Part 1 paper and its Part 2 companion show how SysML supports simulation-based design (SBD) via tutorial-like examples. Our target audience is end users wanting to learn about SysML parametrics in general and its applications to engineering design and analysis in particular. We include background on the development of SysML parametrics that may also be useful for other stakeholders (e.g, vendors and researchers).

In Part 1 we walk through models of simple objects that progressively introduce SysML parametrics concepts. To enhance understanding by comparison and contrast, we present corresponding models based on composable objects (COBs). The COB knowledge representation has provided a conceptual foundation for SysML parametrics, including executability and validation. We end with sample analysis building blocks (ABBs) from mechanics of materials showing how SysML captures engineering knowledge in a reusable form. Part 2 employs these ABBs in a high diversity mechanical example that integrates computer-aided design and engineering analysis (CAD/CAE).

The object and constraint graph concepts embodied in SysML parametrics and COBs provide modular analysis capabilities based on multi-directional constraints. These concepts and capabilities provide a semantically rich way to organize and reuse the complex relations and properties that characterize SBD models. Representing relations as non-causal constraints, which generally accept any valid combination of inputs and outputs, enhances modeling flexibility and expressiveness. We envision SysML becoming a unifying representation of domain-specific engineering analysis models that include fine-grain associativity with other domain- and system-level models, ultimately providing fundamental capabilities for next-generation systems lifecycle management.

**Keywords.** SysML parametrics, composable object (COB), constraint graph, non-causal, multi-directional, simulation-based design, design-analysis integration, CAD-CAE interoperability.

## 1 Introduction to SysML and COBs

The purpose of these papers is to introduce SysML parametrics technology in general and its application to engineering analysis and design in particular. Composable object (COB) technology is included as one possible means to execute SysML parametric models. This section overviews these technologies and assumes the reader is familiar with basic object-oriented concepts and engineering analysis.

### 1.1 OMG SysML™ and Parametrics

The official Object Management Group (OMG) website describes SysML as follows [OMG, 2007a]:

The OMG Systems Modeling Language (OMG SysML™) is a general-purpose graphical modeling language for specifying, analyzing, designing, and verifying complex systems that may include hardware, software, information, personnel, procedures, and facilities. In particular, the language provides graphical representations with a semantic foundation for modeling system requirements, behavior, structure, and integration with a broad range of engineering analysis. SysML represents a subset of UML2 with extensions needed to satisfy the requirements of the UML™ for Systems Engineering RFP. SysML uses the OMG XML Metadata Interchange (XMI®) to exchange modeling data between tools, and is also intended to be compatible with the evolving ISO 10303-233 systems engineering data interchange standard.

<sup>†</sup> Corresponding author: [Russell.Peak@gatech.edu](mailto:Russell.Peak@gatech.edu)

See the [GIT, 2007c] website for potential updates to this material.

The UML for Systems Engineering RFP was developed jointly by the OMG and the International Council on Systems Engineering (INCOSE) and issued by the OMG in March 2003. The RFP specified the requirements for extending UML to support the needs of the systems engineering community. The SysML Specification was developed in response to these requirements by a diverse group of tool vendors, end users, academia, and government representatives. The OMG announced the adoption of the OMG SysML™ on July 6, 2006.

Unless otherwise noted, the diagrams and description in these papers (Parts 1 and 2) conform to the SysML Final Adopted Specification plus changes proposed by the SysML Finalization Task Force released February 23, 2007 [OMG, 2007b]. Additional information on OMG SysML publications, tutorials, vendor tools, and links to other items mentioned in this subsection can be found via <http://www.omgsysml.org/>, including [Burkhart, 2006] and [Friedenthal *et al.* 2006].

**Motivation for SysML parametrics—Part 1.** Parametrics are not part of UML and were introduced as a new modeling capability as part of SysML (Figure 11). A primary focus for parametrics is to support engineering analysis of critical system parameters including evaluation of performance, reliability, and physical characteristics. Engineering analysis is a critical aspect of systems engineering. However, there has been a fundamental gap in previous modeling languages including UML, IDEF, and behavior diagrams. As a result, non-standardized engineering analysis models are often disjoint from the system architectural models that specify the behavioral and structural aspects of a system. The lack of integration and synchronization between the system architectural models and the engineering analysis models is aggravated as the complexity, diversity, and number of engineering analysis models increases. Parametrics provides a mechanism to address this gap and integrate engineering analysis models with system requirements and design models for behavior and structure. In addition, parametrics is a constraints representation that can be used more generally to capture other types of knowledge beyond support for engineering analysis. Part 2 provides further motivation related to parametrics and engineering analysis in particular.

Key SysML parametrics terminology includes the following:

- A *constraint* is analogous to an equation (e.g.,  $F=m*a$ , which is useful in many engineering problems). A *constraint block* is a definition of this equation in a way that it can be reused. A *constraint property* is a particular usage of a generic constraint block (e.g., to support engineering analysis of a specific design).
- A *parameter* is a variable of an equation (a constraint) such as  $F$ ,  $m$ , and  $a$  in the above example.
- A *value property* is any quantifiable characteristic of the environment, system, or its components that is subject to analysis such as the mass of a system or its components. The basic properties of a system architectural model and its components are typically represented in SysML models as value properties.

Value properties are bound to the parameters of a constraint. This binding is the mechanism to keep the generic equations, such as  $F=m*a$ , linked to the value properties that are used to specify the system and its components. The complete terminology includes additional terms and concepts, but the above are the essential elements to keep in mind as you begin to work with SysML parametrics.

## 1.2 The Composable Object (COB) Knowledge Representation

SysML parametrics is based in part on a theory called composable objects (COBs). Composable objects have been developed at the Georgia Institute of Technology (GIT) as a means for representing and integrating design models with diverse analysis<sup>1</sup> models. Design and analysis information is typically represented by a collection of interrelated models of varying discipline and fidelity. Thus a method for capturing diverse multi-fidelity models and their fine-grained relations was needed. It was also desirable for this method to be independent of the specific CAD/CAE tools used to create, manage, and compute these models.

The COB representation is based on object and constraint graph concepts to gain their modularity and multi-directional capabilities. Object techniques provide a semantically rich way to organize and reuse the complex relations and properties that naturally underlie engineering models. Representing relations as constraints makes COBs flexible because constraints can generally accept any combination of I/O information flows. This multi-directionality enables design sizing and design verification using the same COB-based analysis model. Engineers perform such activities throughout the design process, with the former being characteristic of early design stages and vice versa.

The COB representation includes several modeling languages (see Appendix A). It has lexical formulations that are computer interpretable, as well as graphical forms that aid human comprehension (Figure 14). Two lexical languages, COS and COI, are the master forms which are both computer-interpretable and human-friendly. Other

---

<sup>1</sup> In this overview the terms “analysis” and “simulation” synonymously denote modeling physical behavior such as stress and temperature. Envisioned next-generation extensions include generalizations for broader classes of modeling and simulation.

forms depict subsets of COS and COI models. For example, the graphical constraint schematic notation (Figure 15) emphasizes object structure and relations among object variables and has strong electrical schematic analogies. The structure level languages define concepts as templates at the schema level, whereas the instance level defines specific objects that populate one or more of these templates.

Over the past few years we have worked together with other SysML developers to embody COB concepts within SysML (especially regarding its block and parametric constructs) [GIT, 2007c; Peak, 2002; Peak *et al.* 2005a, 2005b]. This approach has benefited both SysML and COBs—the former by providing conceptual formalisms and a broad variety of examples; and the latter by leveraging a richer set of SysML/UML2-based constructs that integrate with other system specification and design features, and by providing graphical modeling capabilities via multiple commercial tool vendors.

## 2 Introductory Concepts and Tutorial Examples

This section introduces SysML and COB concepts using tutorial-like examples. COBs are leveraged here as a preexisting body of work that illustrates typical parametrics concepts. The intent is to help the reader better understand such concepts as used in SysML parametrics. Over time we intend to convert these examples to fully use SysML as their content representation (e.g., for all graphical modeling views), while continuing to employ COB constraint management algorithms for parametrics execution. See [GIT, 2007c], [Peak *et al.* 1999, 2002, 2005a, 2005b] and [Wilson 2000, 2001] for further information and examples.

### 2.1 Triangles and Prisms

**Representation Using Classical COB Formulations.** Figure 1 provides the main classical COB formulations for a primitive object (a right triangle) and its usage in a compound object (a prism). The upper-left portion of Figure 1 shows two forms traditionally used to convey knowledge about such objects: equations that hold in the object and a corresponding graphical view depicting variables in these equations. In COB terms, we call these *relations-S* and a *shape schematic-S*, respectively. The -S denotes that the representational construct describes a structural facet of the object—a class or schema-level facet in object-oriented terms—which can be thought of as a kind of template.

Representation of the first object as a COB *right\_triangle* template is shown in Figure 1.1(a)-(e), where the *constraint schematic-S* form, (c), depicts its equation-based relations and variables as a kind of graph (Figure 15). Figure 1.1(e) is the *lexical COB structure (COS)* form, which is executable. It is also the master template from which structural forms (b)-(d) can be derived. *Subsystems*, (d), are encapsulated views that represent this object when it is used as a building block in other COBs.

Structural forms of a COB *triangular\_prism* template are similarly shown in Figure 1.2. Besides basic variables like volume and length, this template has one compound variable, cross-section, which is a usage of the *right\_triangle* template defined above.

The equation relations in all the COB structural forms are non-causal—i.e., they are undirected and support any valid input/output combination at the instance level. For example, relation *r1* in *right\_triangle*, can have variables *base* and *height* as inputs to produce *area* as the output, or *area* and *height* can be inputs to produce *base* as the output.

A COB instance can be loosely thought of as a copy of a COB template that has values filled in and input/output directions specified. Figure 1.2(f) is the *constraint schematic-I* form of a *triangular\_prism* instance after it has been solved. Figure 1.2(g) shows the corresponding *lexical COB instance (COI)* form for state 1.1, as well as the form for state 1.0 (before solving). Variables *cross-section.base*, *cross-section.height*, and *length* are provided as inputs in these states, and the primary target to be solved for is *volume*. As seen in state 1.1, note that *cross-section.area* was produced as an intermediate (ancillary) output along the way.

**Representation Using SysML.** The same objects can be represented using similar concepts in SysML, as seen in the Figure 2 SysML diagrams. In brief, COBs are represented using SysML *blocks*, primitive variables are called *value properties*, and SysML *parametric diagrams* essentially replace COB constraint schematics. SysML represents relations themselves as a special type of SysML block known as a *constraint block*, which elevates their semantics and makes them explicitly reusable (vs. their simple string form in COS). Usages of constraint blocks are known as *constraint properties* and are shown as rounded corner rectangles (e.g., *r1* and *r2* in *RightTriangle*).

Figure 2(a) is a *block definition diagram (bdd)* that shows a block called *RightTriangle* that includes value properties named *area*, *base*, *diagonal*, and *length*. Block definition diagrams present structural relationships among blocks based on concepts including *part-of* and *is-a* relationships. This type of diagram is based on the UML2 *class diagram* and essentially replaces the classical COB *object relationship diagram-S* (Figure 14). The value properties represent quantifiable characteristics that can include units and dimension such as units of meters and dimension of

length. This is accomplished in SysML by typing the value property by a *value type*. In Figure 2(a), the value types are shown as LengthMeasure, AreaMeasure, and VolumeMeasure. These represent abstract value types that users can employ to make their model libraries usable with a wide variety of unit systems. Appendix A (Figure 12) defines these value types and sample concrete specializations that are used throughout Parts 1 and 2.

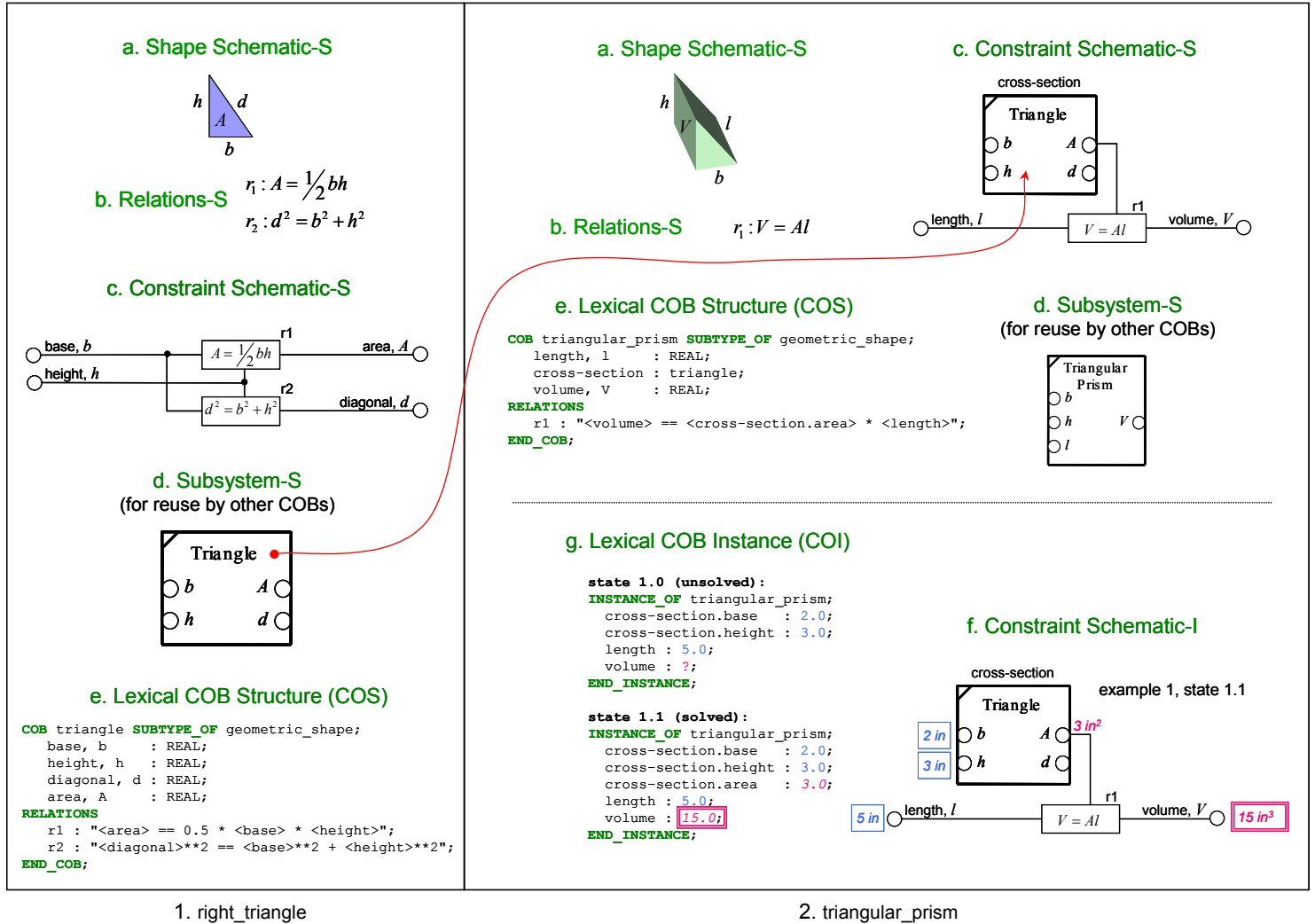


Figure 1: Triangle/prism tutorial—classical COB formulations.

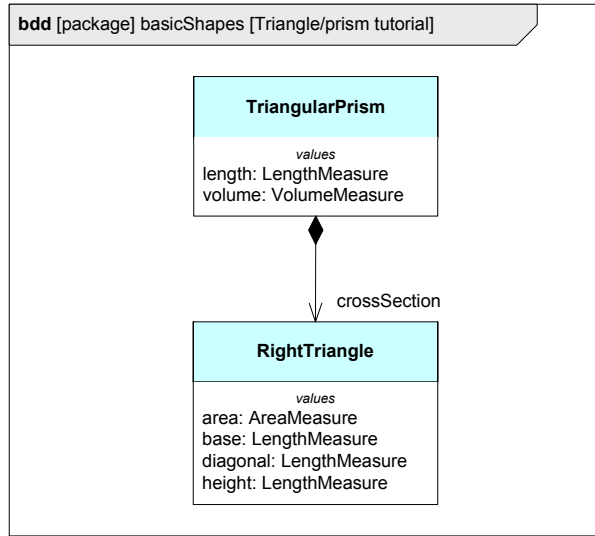
Figure 2(b) is a *parametric diagram (par)* that shows how these value properties are constrained by area equation and Pythagorean equation usages (i.e. constraint properties). This is accomplished by binding the parameters of the equations to the value properties of the RightTriangle block, which are depicted by the small rectangles in the figure. This binding is achieved using *binding connectors* which are graphically represented as solid lines<sup>2</sup> that “wire together” value properties and parameters. Appendix A contains a sample SysML tool screenshot (Figure 13) where such models have been implemented based on earlier drafts of the SysML spec.

Figure 2(a) also defines a TriangularPrism block which is composed of a RightTriangle and has two value properties. The particular usage of the RightTriangle for TriangularPrism is referred to by its part name called

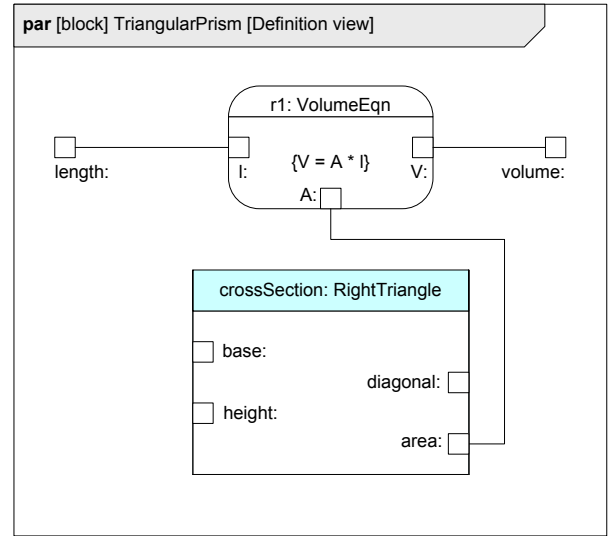
<sup>2</sup> The parametric diagrams in these papers use an electrical schematic-like branching convention for connecting a value property to more than one constraint parameter (e.g., height connecting to both r1.h and r2.h in Figure 2(b)). This branching approach reduces the graphical space needed to connect to a value property and typically results in a cleaner looking diagram. The convention for unambiguously interpreting such branching is that every connection between the value property and the indicated parameters is implemented as a distinct binding connector. Thus two binding connectors attach to height in Figure 2(b). This convention applies similarly for the case where a parameter connects directly to two or more other parameters.

crossSection. Figure 2(c) is a parametric diagram that further defines the TriangularPrism block by showing how the length, volume, and crossSection.area value properties are constrained by the r1 volume equation.

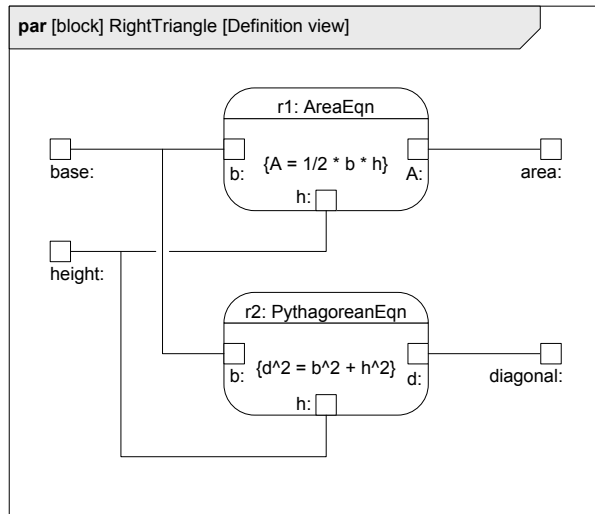
Figure 2(d) shows a particular instance<sup>3</sup> of the constrained properties from Figure 2(c) that includes specific values for the value properties. It represents the same COB instance shown in Figure 1 and uses a similar color coding scheme<sup>4</sup> to depict the current inputs (in blue) and outputs (in red). This example utilizes value types based on millimeters. SysML syntax places what looks like the units to the left of the value because these “units” are actually concrete value types. For example, mm^2 is a specialization of AreaMeasure as defined in Figure 12.



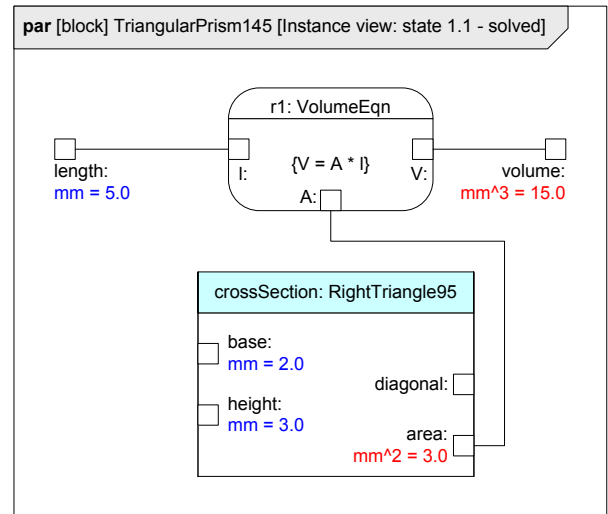
(a) Triangle/prism tutorial block definition diagram.



(c) TriangularPrism parametric diagram.



(b) RightTriangle parametric diagram.



(d) TriangularPrism sample instance.

Figure 2: Triangle/prism tutorial—SysML diagrams.

There is an important distinction between a constraint block that defines a constraint, and a constraint property, which is a usage of a constraint block. A constraint block definition can represent a generic equation, such as the volume, area, and Pythagorean equations. Constraint blocks may be reused to constrain value properties of many

<sup>3</sup> The term *instance* is loosely used its generic form. Briefly, we employ an “instantiation by specialization” approach such that RightTriangle95 is a specialization that has specific values. Specifics of this approach are beyond the scope of these papers.

<sup>4</sup> This causality color scheme is a GIT user/tool convention.

different contexts such as the example shown in Figure 2. The colon notation distinguishes the usage from the definition. In Figure 2(b) for example, the heading `r1:AreaEq` indicates that `r1` is a specific usage of the generic constraint block definition called `AreaEq`.

In general, we envision that analysis libraries representing concepts and reusable equations for specific analysis domains such as fluid dynamics will be implemented as SysML blocks and constraint blocks. The remaining examples in Part 1 demonstrate one way to capture such knowledge. Part 2 shows how these analysis libraries can be readily used to bind the value properties of the system design/architectural models to various types of analysis.

**Implementation and Execution.** SysML is intended to capture actual parametric models—not just documentation—which can be solved by external tools. Here we describe one way to achieve this executable intent.

As seen in Figure 13, vendors are beginning to support SysML in their modeling tools—often by leveraging their existing UML2 capabilities. Underneath the graphical diagrams and model tree browsers, these tools typically capture model structure in a semantically rich manner. This enables ready execution of SysML constructs based on UML2 such as sequence diagrams. However, since parametric constructs are new, current tools typically do not support parametrics execution.

*XaiTools*<sup>TM</sup> [Wilson et al. 2001; Peak, 2000] is a toolkit that implements practically all COB concepts except the graphical forms. Originally developed at GIT for design-analysis integration, it imports and exports COB models in various lexical formats, including COS and COI files (Figure 15). It includes constraint graph management algorithms that determine when to send what relations and values to which solvers, and how to pass the subsequent results to the rest of the graph. In this way *XaiTools* enables links with design tools and effectively provides an object-oriented, constraint-based, front end to traditional CAE tools, including math tools like *Mathematica* and finite element analysis tools like *Abaqus* and *Ansys* (Figure 16).

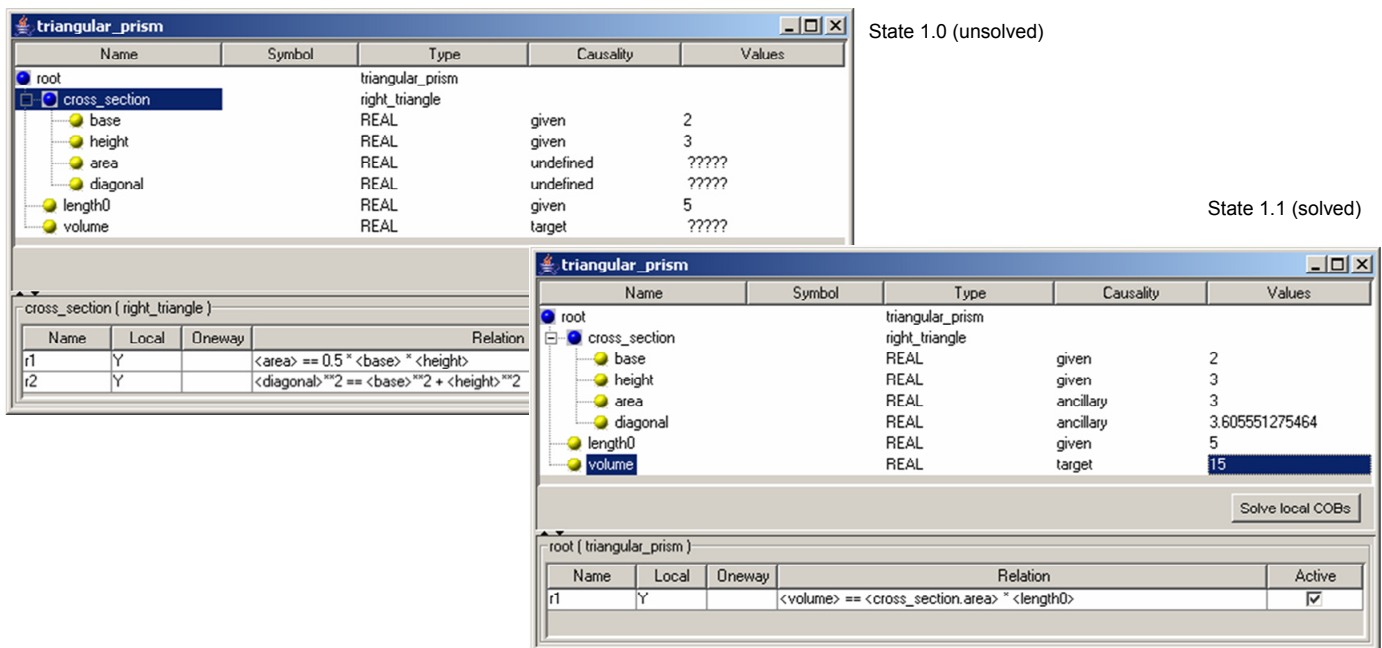


Figure 3: TriangularPrism instance (from Figure 2) in *XaiTools* COB Browser—an object-oriented non-causal spreadsheet.

Leveraging these capabilities, we implemented the prototype architecture shown in Appendix A (Figure 17) to demonstrate how one can author COBs as parametric models in a SysML tool and then execute them using commercial solving tools. COS and COI models are generated from the SysML model structure and exported as text files. *XaiTools* then loads these models and executes them as usual. Figure 3 shows the triangular prism instance described earlier in before and after solving states. This *XaiTools* COB Browser application can be considered an object-oriented non-causal spreadsheet that captures and manages the execution of parametric models.

## 2.2 Analytical Spring Systems

Next we walk through models of several analytical objects from high school physics. The primary purpose of these



tutorial examples is to introduce additional SysML and COB modeling concepts. It also introduces thought processes regarding how one can go about capturing such knowledge in the form of reusable SysML-based libraries.

**Representation Using Classical COB Formulations.** Figure 4 (a) and (b) show traditional forms of an idealized linear spring object. Its shape schematic identifies variables and their idealized geometric context, and relations define algebraic equations among these variables. As with the triangle/prism tutorial, Figure 4 (c)-(e) provide other structural formulations that define linear\_spring as a COB template.

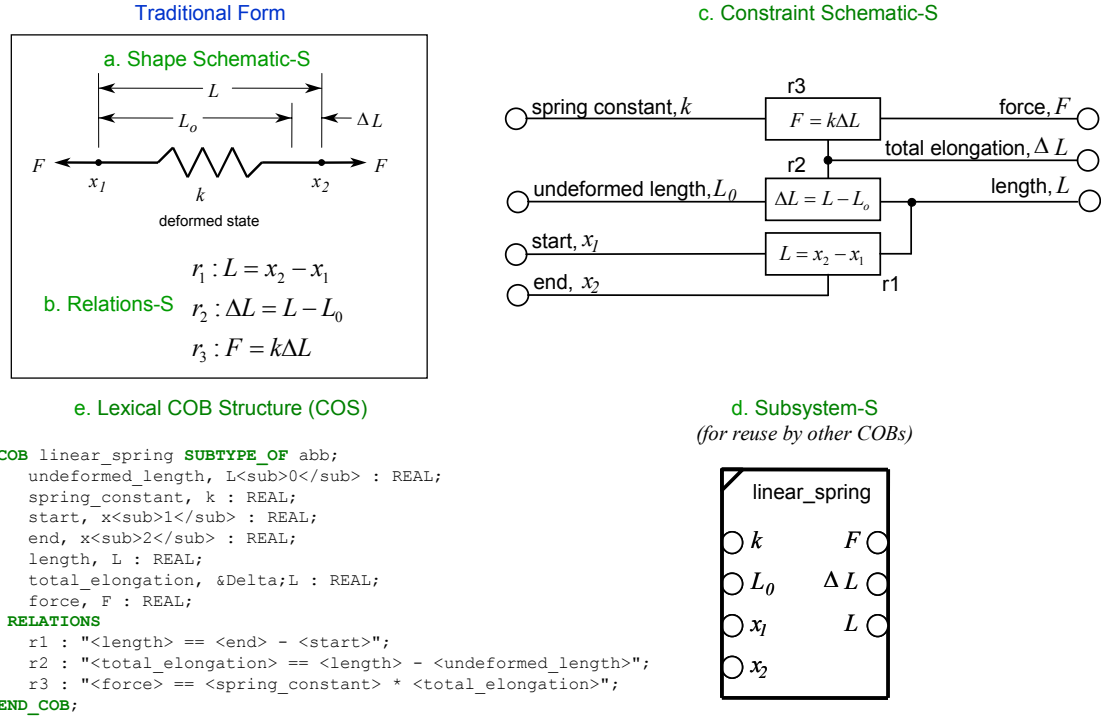


Figure 4: Analytical springs tutorial—linear\_spring classical COB structural formulations.

Figure 5 shows a linear\_spring instance in two main states. Throughout state 1, spring\_constant, undeformed length, and force are the inputs, and total\_elongation is the desired output. The COI form, (b), represents state 1.0 as this COB instance exists before being solved. State 1.1 shows it after solution, where one can see that length was also computed as an ancillary result. Variables end and start have no values in State 1.1 because sufficient information was not provided to determine them. State 5 shows this same linear\_spring instance later after the user changed the length variable to be an input, provided it a length value (the desired length after deformation), and changed the spring\_constant variable to be the target output.

Considering engineering semantics behind such scenarios, one sees that state 1 typifies a kind of design verification scenario—the "natural inputs" (i.e., key design-oriented properties and a load) are provided as inputs and a "natural output" (i.e., deformed length—a response to the load) is the requested output. Hence, the analytical spring "design" is being checked to ensure it gives the desired response.

Along these lines, state 5 is a kind of simple design synthesis (sizing) scenario. It reverses the situation by making one natural output into an input and setting one natural input to be the target output. It effectively asks "what spring\_constant (a design-oriented variable) do I need to achieve the desired deformed length (a response)?" This COB capability to change input and output directions with the same object instance can be applied to more complex situations. It is a multi-directional capability in that there are generally many possible input/output combinations for a given constraint graph structure. It enables a form of simulation-based design (SBD) by driving design from simulation and vice-versa.

Next we look at how systems can be composed from basic elements like linear\_spring. Given an analytical system containing two idealized springs as in Figure 6(a1), in traditional modeling approaches one might start by drawing their freebody diagrams, (a2). From there one could list the analytical spring formulae and boundary conditions, (b), and solve the resulting set of equations for target outputs (e.g., elongations at the individual and

system levels). One could use symbolic math tools like *Maple* or *Mathematica* to aid this process and change input/output combinations. However, one would essentially end up having a list of equations with only implied engineering semantics. For example, one could not query relation  $r11$  in the list and directly know that it is part of the first spring. Furthermore, adding and deleting equations to change input/output directions for a large system of equations could become unwieldy.

When one considers the constraint graph (Figure 6(d)) for this spring system, one recognizes that the shaded portions are essentially duplications of the same kind of relations (e.g.,  $r11$  vs.  $r21$ ). Traditionally, one would have to manually replicate and adjust these similar relations via a tedious and error-prone process. COBs address these semantic and usability issues by grouping relations and variables according to their engineering meaning and capturing them within explicit reusable entities.

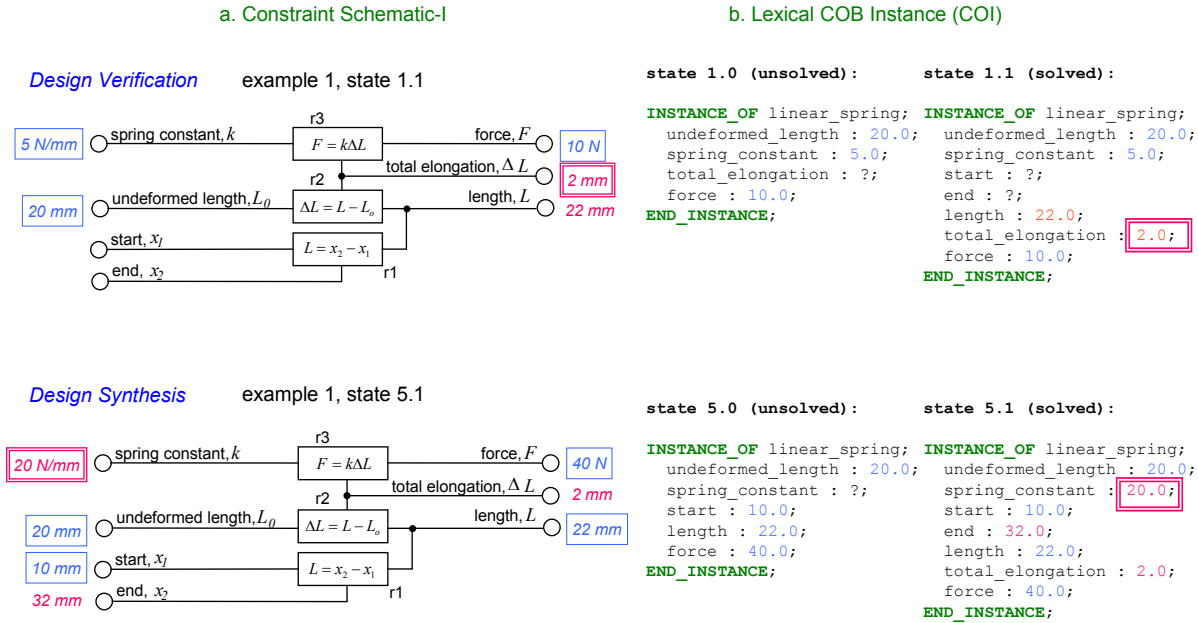


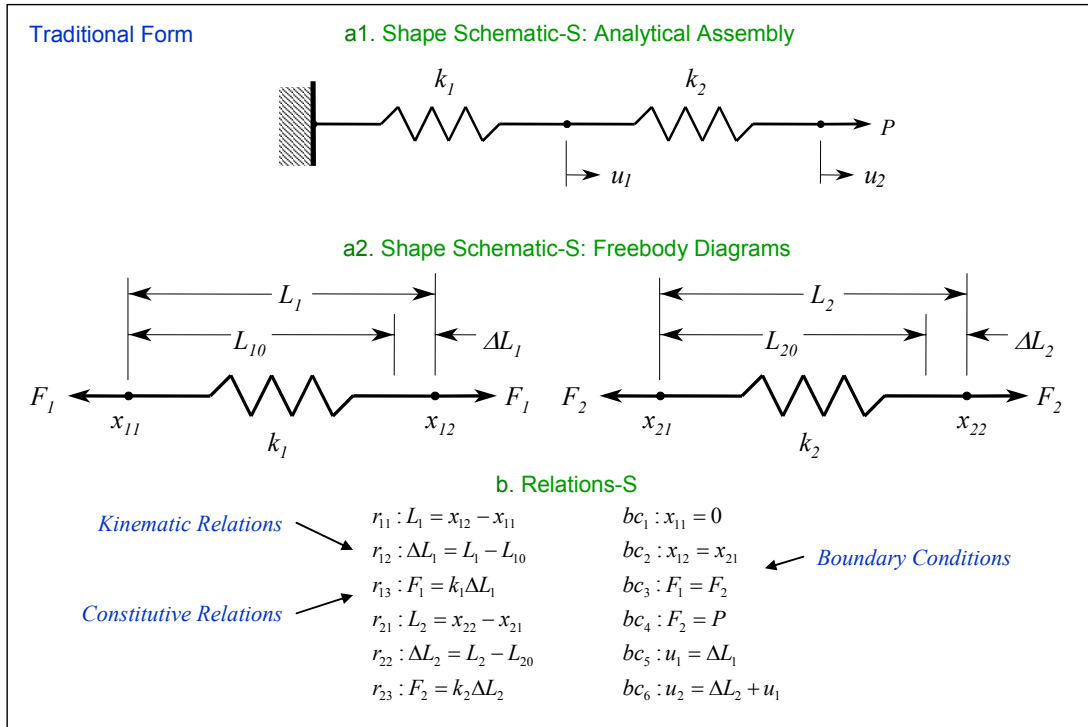
Figure 5: Multi-directional (non-causal) capabilities of a COB linear\_spring instance.

For example, by applying object-oriented thinking, the shaded regions in Figure 6(d) can be represented by two linear\_spring subsystems, as is done in (c). There is no need to specify these relations in the corresponding system-level COS form, (e), as they are already included in the linear\_spring entity per its COS definition (Figure 4). System-level relations such as boundary conditions are the only relations that need to be specified here. With this definition completed, the constraint graph can now be seen as another COB form that is derivable from the lexical form. A *constraint graph-S* like (d) is effectively the fully decomposed form of its corresponding<sup>5</sup> constraint schematic-S (c)—it is flattened such that no subsystem encapsulations are present.

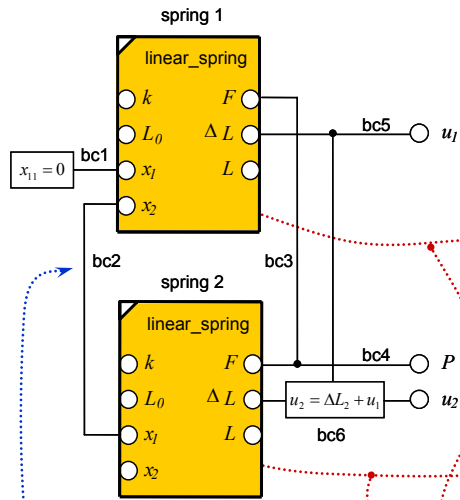
**Representation Using SysML.** One can similarly render SysML models for this analytical springs tutorial. Figure 7(a) is a block definition diagram showing that TwoSpringSystem is composed of two *part properties*, named spring1 and spring2, that utilize LinearSpring. Figure 7(b) and (c) show SysML parametric diagrams for LinearSpring and TwoSpringSystem respectively. Note again the richer semantics enabled by SysML value types. DistanceMeasure values are intended to represent position distance in space relative to an origin and can take on any real number, while LengthMeasure values represent shape measures of physical things and thus they must be non-zero positives.

<sup>5</sup> COBs can also have elements such as options, conditional relations, and variable length aggregates, in which case they have not one but a family of corresponding constraint graphs.

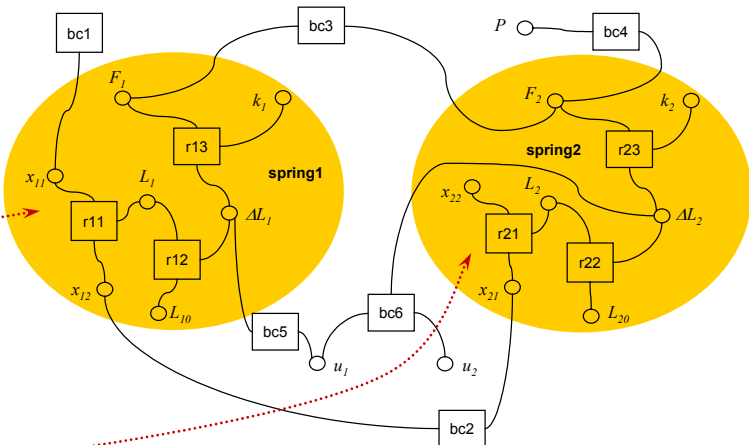




**c. Constraint Schematic-S**



**d. Constraint Graph-S**



**e. Lexical COB Structure (COS)**

```

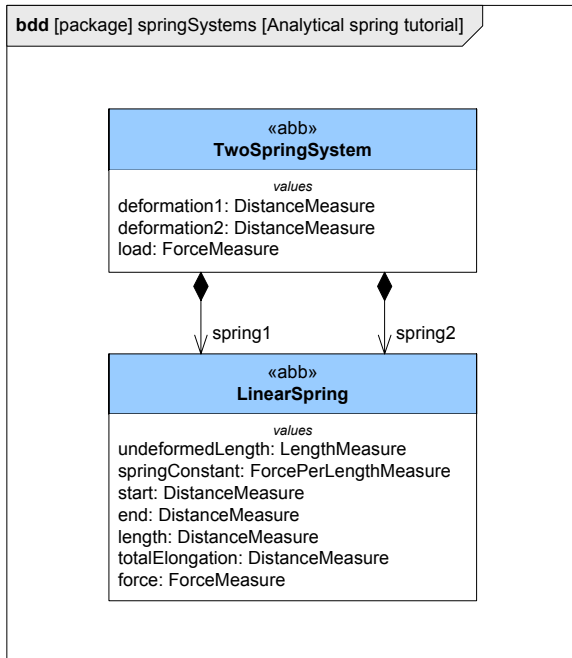
COB two_spring_system SUBTYPE_OF analysis_system;
spring1 : linear_spring;
spring2 : linear_spring;
deformation1, u<sub>1</sub> : REAL;
deformation2, u<sub>2</sub> : REAL;
load, P : REAL;

RELATIONS
bc1 : "<spring1.start> == 0.0";
bc2 : "<spring1.end> == <spring2.start>";
bc3 : "<spring1.force> == <spring2.force>";
bc4 : "<spring2.force> == <load>";
bc5 : "<deformation1> == <spring1.total_elongation>";
bc6 : "<deformation2> == <spring2.total_elongation> + <deformation1>";

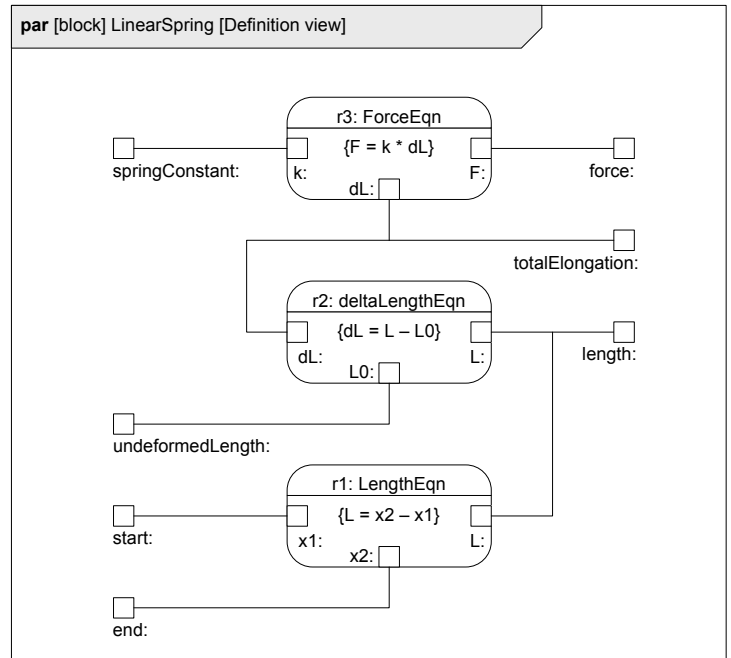
END_COB;

```

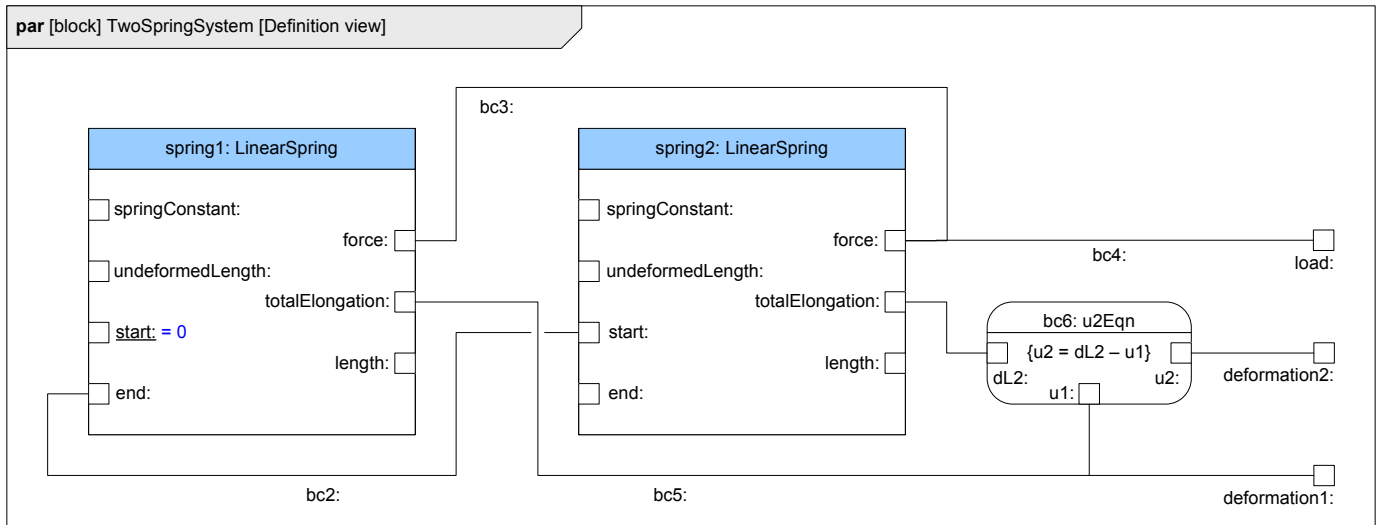
Figure 6: Analytical springs tutorial—two\_spring\_system classical COB structural formulations.



(a) Analytical springs tutorial block definition diagram.

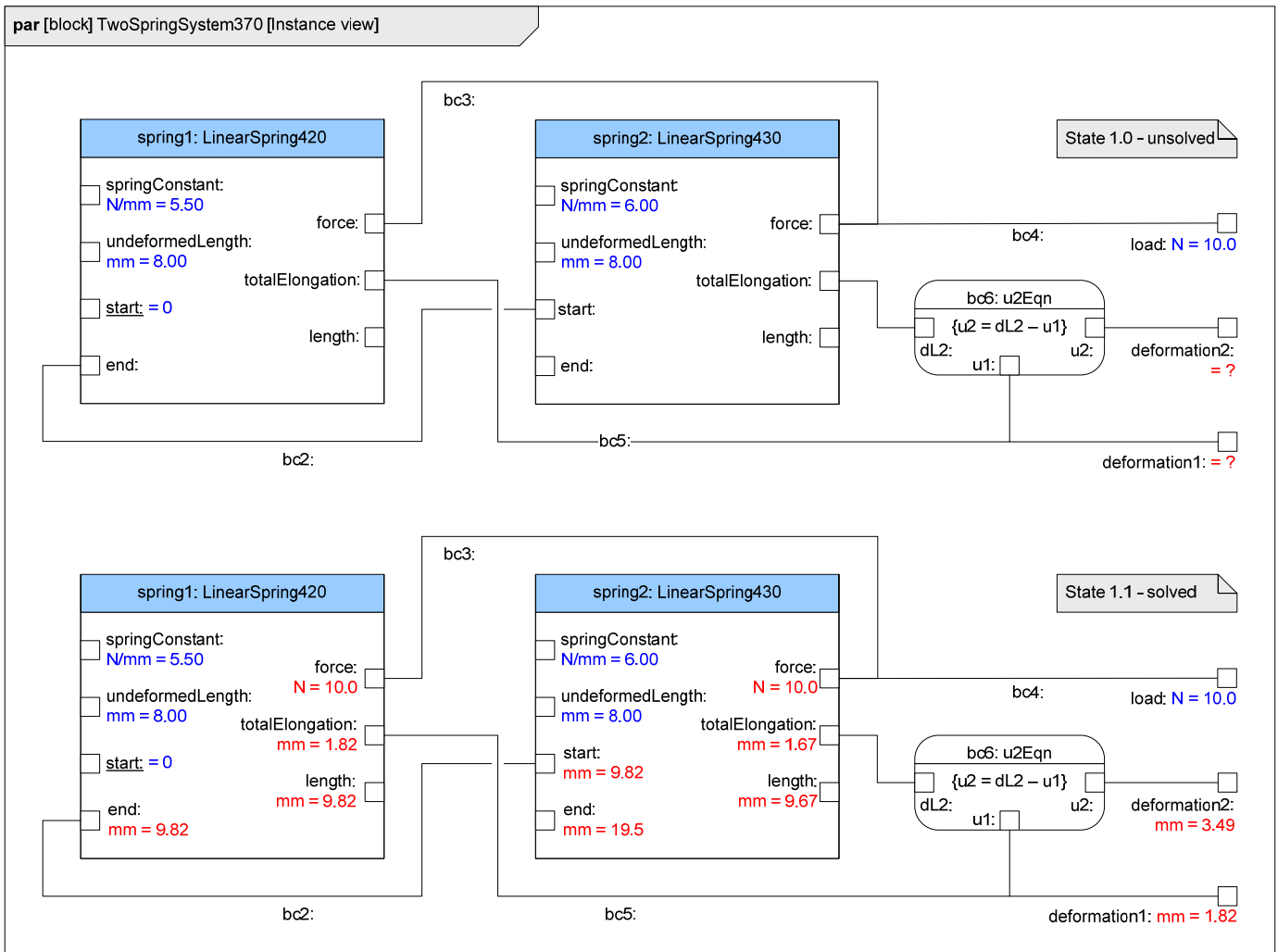


(b) LinearSpring parametric diagram.



(c) TwoSpringSystem parametric diagram.

Figure 7: Analytical springs tutorial—TwoSpringSystem SysML diagrams.

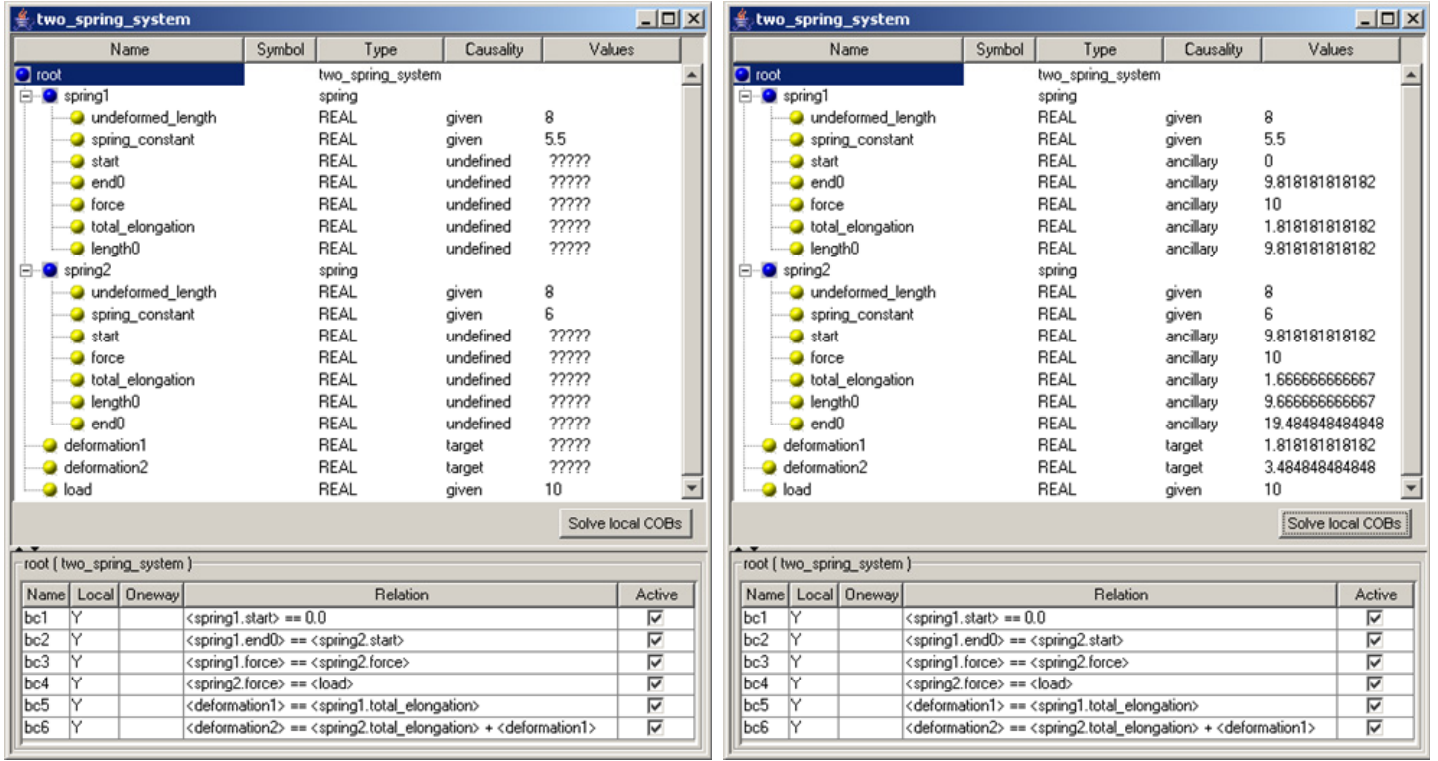


(d) TwoSpringSystem parametric diagram – sample instance.

Figure 7 (continued): Analytical springs tutorial—TwoSpringSystem SysML diagrams.

Figure 7(d) represents an instance of TwoSpringSystem with state 1.0 being the inputs (givens) for a kind of system design verification scenario. The user provides input values (in blue) for the intrinsic characteristics of the analytical springs and a system load, and also indicates the primary target properties to solve for (deformation1 and deformation2) as denoted by the red question marks. State 1.1 shows the results after solving, where one sees that the total system deformation ( $u2$ ) has been determined to be 3.49 mm. At this point the user could change one of the input values and resolve (e.g., increasing spring1.springConstant to reduce the total deformation). Or one might change input/output directions and calculate the required spring constant to achieve the target total deformation.

**Implementation and Execution.** Figure 8(a) is lexical COB instance corresponding to Figure 7(d) state 1.0 as generated from a parametrics model in a SysML tool (using the approach outlined in Figure 17). CXI is a relatively newer COB lexical that is essentially an XML-based form of COI plus new capabilities (e.g., capturing causality more completely). *XaiTools* reads in this CXI model and displays it in a COB browser before and after solving. These figures exemplify the complementary relationship between SysML parametrics (for model authoring and management) and COBs (for model execution).



(a) Lexical  
COB instance  
as XML (CXI)

example 2,  
state 1.0 (unsolved)

```
<linear_spring loid="_15">
  <undeformed_length causality="given">8.0</undeformed_length>
  <spring_constant causality="given">5.5</spring_constant>
</linear_spring>

<linear_spring loid="_25">
  <undeformed_length causality="given">8.0</undeformed_length>
  <spring_constant causality="given">6.0</spring_constant>
</linear_spring>
```

```
<two_spring_system loid="_3">
  <spring1 ref="_15"/>
  <spring2 ref="_25"/>
  <deformation1 causality="target"/>
  <deformation2 causality="target"/>
  <load causality="given">10.0</load>
</two_spring_system>
```

Figure 8: A COB instance generated from a SysML TwoSpringSystem model (Figure 7d) and its execution in XaiTools.

### 3 Analysis Building Block Examples

SysML can be used to represent analytical engineering concepts as analysis building blocks (ABBs) [Peak, 2000; Peak *et al.* 2001]. This section highlights sample ABBs representing mechanics of materials domain knowledge.

Figure 9(a) is the SysML parametric diagram for a material behavior ABB known as OneDLinearElasticModel. This analytical model includes material properties such as Young's modulus and relations such as stress-strain-temperature constitutive equations. It is typically re-used in building continuum primitives like the two shown in the same figure: (b) ExtensionalRod and (c) TorsionalRod. These continuum primitives in turn can be used to build other analysis models, including product-specific analysis models as seen in Part 2 [Peak *et al.* 2007].

Note that traditional relations like Equation 1 (for total deformation in an extensional rod) need not be explicitly included in parametric models. Rather, these types of relations are derivable from the fundamental relations present in their corresponding ABB blocks; thus, the effective behavior of Eqn. 1 is automatically included in ExtensionalRod.

$$\Delta L = \frac{FL}{EA} + \alpha \Delta TL$$

Equation 1

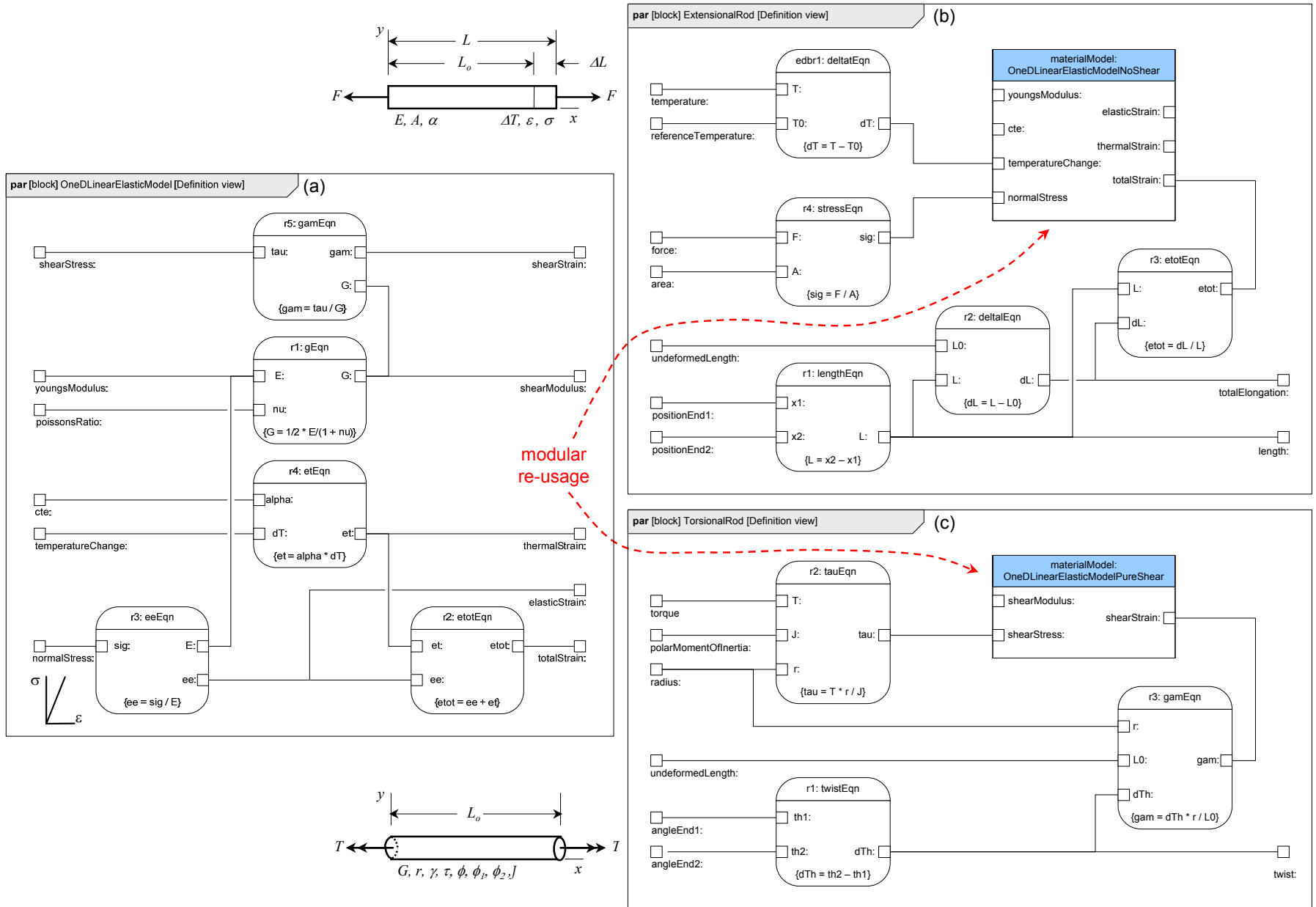


Figure 9: Example mechanics of materials analysis build blocks (ABBs)—SysML parametric diagrams.

## 4 Discussion

Normally end users will not need to create from scratch primitive templates like those above—triangle, prism, linear spring, 1D material model, extensional rod, and torsional rod. Such commonly used entities should be pre-defined and available in ready-to-use SysML libraries, along with frequently occurring ABB systems. This approach will empower users to concentrate on composing higher-level models to represent and solve their engineering analysis problems. However, it will take time and effort to create such libraries. While various vendor-specific engineering model libraries are available<sup>6</sup>, SysML offers a way to wrap existing libraries in a vendor-neutral form and to create new libraries from fundamental principles. Burkhart, Paredis, and colleagues [GIT, 2007c] are pursuing a multi-aspect library approach for fluid power components. Peak [2002b and 2003] provides an initial characterization of such libraries, estimating that there are, for example, on the order of several hundred ABB concepts in the structural analysis domain.

The above examples demonstrate how SysML parametrics has largely subsumed COB graphical modeling capabilities. As one becomes more familiar using a SysML modeling tool, other SysML capabilities beyond those in COB formulations become evident (e.g., stereotypes, value types, and automated consistency checks), and interactively composing diverse models becomes more of a reality.

The above examples also show COB formulation capabilities that are not yet readily available in SysML itself and/or associated tools. Compare, for example, Figure 4(c) with Figure 7 and note that constraint blocks could use a way to show their relations in a localized usage manner (and also offer the option to use symbols)—e.g., representing  $F=k\Delta L$  using a generic  $a=b*c$  relation for  $r3$ . Shape schematics and flattened constraint graphs like Figure 6(a) and (d) are other COB views that may be worth considering for future versions of SysML.

Instances, causality, and human-friendly lexical forms (e.g. XML in Figure 8) are other areas that may warrant further attention, as seen by comparing Figure 1(g)-(f) with Figure 2(d). Frequent interaction with instances (and associated instance libraries) may be one of the key differences between traditional UML2 usage for software development vs. envisioned SysML usage for interactive system design, modeling, and simulation.

Finally, as users begin to get familiar with SysML, it is helpful to keep in mind that SysML is in its infancy, relatively speaking, as a v1.0 technology. Additional capability needs are bound to be identified as the user experience base grows over time. Some may require a v1.1 specification update, and others may not be addressed until v2.0 or beyond. However, as evidenced by this paper and Part 2, experience to date indicates there is good value to be gained utilizing SysML v1.0.

## 5 Summary and Conclusions

This paper progressively introduces SysML parametrics concepts through models of simple objects to demonstrate how engineering analysis models can be integrated with system architecture/design models. Sample analysis building blocks (ABBs) demonstrate how SysML captures engineering knowledge from mechanics of materials. A high diversity example in Part 2 [Peak *et al.* 2007] reuses these ABBs in simulation-based design (SBD) scenarios involving mechanical computer-aided design and engineering (CAD/CAE) models.

This paper also overviews the composable objects (COBs) knowledge representation that a) provided motivation, semantics, and examples during the development of SysML parametrics, and b) continues to provide means for executing SysML parametrics. We presented classical COB formulations alongside SysML diagrams for the same models. This approach has hopefully given the reader a better understanding of SysML parametrics and how to use it to create executable parametric models. It has also highlighted the following mutual benefits.

Benefits from COB technology for SysML parametrics:

- Provides a conceptual foundation
  - Declarative knowledge representation (non-causal)
  - Combination of object and constraint graph techniques
- Contributes engineering analysis test cases
  - Variety: domains, CAD tools, behaviors, fidelities, CAE tools, etc. (see Part 2)
- Exercises and verifies numerous constructs systematically
- Enables architectures and methods for parametrics executability

---

<sup>6</sup> See, for example: <http://www.mathworks.com/applications/controldesign/>, <http://www.modelica.org/library/>, and <http://www.wolfram.com/products/applications/mechsystems/>



Benefits from SysML technology for COBs:

- Provides richer modeling constructs
  - Reusable relations, stereotyping, extended object constructs, etc.
- Integrates with other specification and design features of SysML
- Broadens and enhances support by commercial tools
- Increases modeling effectiveness (leveraging capabilities supported by numerous vendor tools)
  - Tool-aided graphical view creation (e.g., all the SysML diagram types)
  - Automated consistency between views

Test cases show that COBs and SysML together provide these advantages over traditional engineering analysis representations:

- Richer semantics and greater expressivity
- Capture of knowledge in a modular, reusable manner
- Greater solution control

We believe SysML is poised to become a unifying representation for both domain-specific engineering analysis models and system-level models, and to thus enhance multidisciplinary communication and speed system development. Richer knowledge management becomes feasible, including capturing fine-grain associativity among diverse models and leveraging this associativity for model execution. Therefore, we foresee how SysML and COB-like representations might ultimately provide fundamental advances for next-generation systems lifecycle management. This vision is not as farfetched as it might sound considering evidence like Figure 10 and the following quote from the coming generation of system engineers:

*"I believe that this process will be helpful to others because I have been doing the same thing in my head to organize and understand the different equations and to help me solve problems successfully."*<sup>7</sup>

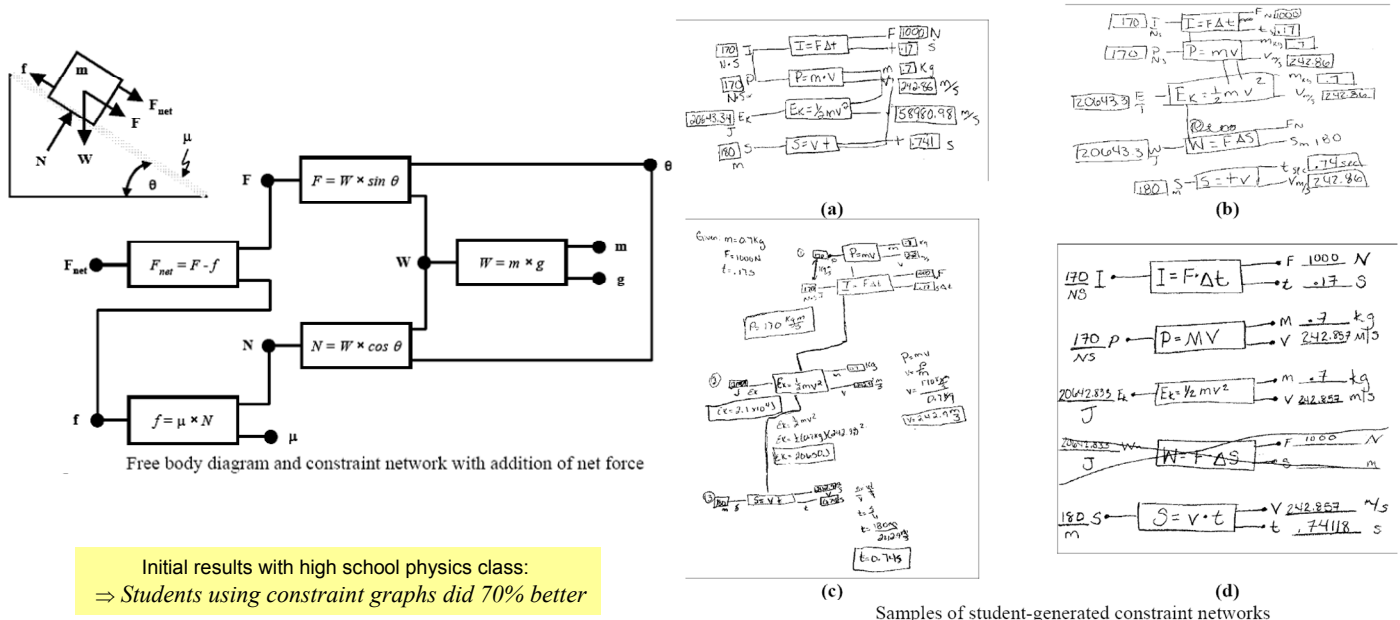


Figure 10: Enhancing education using constraint graph-based knowledge representations. [Cowan et al. 2003]

<sup>7</sup> Source of quote: A student in a high school physics class commenting on using constraint graphs similar to SysML parametrics to aid both concept learning and problem solving [Cowan et al. 2003]. Students who chose to use constraint graphs in this class did 70% better on average than students who stuck with traditional approaches.

## Acknowledgements

This work was funded in part by NASA [GIT, 2007a] and NIST [GIT, 2007b]. SysML modeling work by Andy Scott and Bennett Wilson at GIT has contributed to these examples. While at Artisan Software, Alan Moore played a key role in the development of SysML parametrics, including aiding the incorporation of COB concepts into the specification, as well as their implementation in Artisan Studio. While on the research faculty at Georgia Tech (2005-2006), Diego Tamburini implemented some of the above examples in SysML tool prototypes. He also developed the executable interface for SysML parametrics between Artisan Studio and *XaiTools* mentioned in Appendix A. Similarly, EmbeddedPlus supported GIT efforts to develop an interface to their tools for COB-based solving. Finally, our colleagues in the Georgia Tech SysML Forum have been a steady source of helpful feedback.

## References<sup>8</sup>

- Burkhart RM (2006) Modeling System Structure and Dynamics with SysML Blocks. Frontiers in Design & Simulation Workshop<sup>9</sup>, GIT PSLM Center, Atlanta.
- Cowan FS, Usselman M, Llewellyn D, Gravitt A (2003) Utilizing Constraint Graphs in High School Physics. Proc. ASEE Annual Conf. & Expo. <http://www.cetl.gatech.edu/services/step/constraint.pdf>
- Friedenthal S, Moore A, Steiner F (2006) OMG Systems Modeling Language (OMG SysML) Tutorial. INCOSE Intl. Symposium, Orlando.
- GIT (2007a) The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs), Project Web Page, Georgia Tech, <http://eislabs.gatech.edu/projects/nasa-ngcobs/>.
- GIT (2007b) Capturing Design Process Information and Rationale to Support Knowledge-Based Design-Analysis Integration, Project Web Page, Georgia Tech, <http://eislabs.gatech.edu/projects/nist-dai/>.
- GIT (2007c) SysML Focus Area, Product & Systems Lifecycle Management Center, Georgia Tech, <http://www.pslm.gatech.edu/topics/sysml/>.
- OMG (2007a) OMG Systems Modeling Language (OMG SysML™) website: <http://www.omg-sysml.org/>.
- OMG (2007b) OMG Systems Modeling Language (OMG SysML™) Specification, <http://www.omg.org/>
- SysML 1.0 Proposed Available Specification (PAS), OMG document [ptc/2007-02-03] dated 2007-03-23.
- Paredis CJJ, Diaz-Calderon A, Sinha R, Khosla PK (2001) Composable Models for Simulation-Based Design. *Engineering with Computers*. Vol. 17, pp. 112-128.
- Peak RS, Scholand AJ, Tamburini DR, Fulton RE (1999) Towards the Ubiquitization of Engineering Analysis to Support Product Design. Invited Paper for Special Issue: Advanced Product Data Mgt. Supporting Product Life-Cycle Activities, *Intl. J. Comp. Applications Technology* 12(1): 1-15.
- Peak RS (2000) X-Analysis Integration (XAI)<sup>10</sup> Technology. EIS Lab Report EL002-2000A, Georgia Tech, Atlanta.
- Peak RS and Wilson MW (2001) Enhancing Engineering Design and Analysis Interoperability—Part 2: A High Diversity Example. 1st MIT Conf. Computational Fluid & Structural Mechanics, Boston.
- Peak RS (2002a) Part 1: Overview of the Constrained Object (COB) Engineering Knowledge Representation. Response to UML for Systems Engineering Request for Information (SE DSIG RFI 1) OMG document ad/2002-01-17. <http://syseng.omg.org/>.
- Peak RS (2002b) Techniques and Tools for Product-Specific Analysis Templates Towards Enhanced CAD-CAE Interoperability for Simulation-Based Design & Related Topics. Intl. Conf. Electronic Packaging. JIEP/ IMAPS Japan, IEEE CPMT. Tokyo.
- Peak RS, Lubell J, Srinivasan V, Waterbury SC (2004) STEP, XML, and UML: Complementary Technologies. Product Lifecycle Management (PLM) Special Issue. *J. Computing & Information Science in Engineering*, 4 (4) 379-390.
- Peak RS, Friedenthal S, Moore A, Burkhart R, Waterbury S, Bajaj M, Kim I (2005a) Experiences Using SysML Parametrics to Represent Constrained Object-based Analysis Templates. NASA-ESA Workshop on Product Data Exchange (PDE), Atlanta. <http://eislabs.gatech.edu/pubs/conferences/2005-pde-peak/>
- Peak RS, Tamburini DR, Paredis CJJ (2005b) GIT Update: Representing Executable Physics-based CAE Models in SysML. Presentation to OMG SE DSIG, Burlingame CA. <http://eislabs.gatech.edu/pubs/seminars-etc/2005-12-omg-se-dsig-peak/>
- Peak RS, Burkhart RM, Friedenthal SA, Wilson MW, Bajaj M, Kim I (2007) Simulation-Based Design Using SysML—Part 2: Celebrating Diversity by Example. INCOSE Intl. Symposium, San Diego.
- Wilson MW (2000) *The Constrained Object Representation for Engineering Analysis Integration*. Masters Thesis, Georgia Tech, Atlanta.
- Wilson MW, Peak RS, Fulton RE (2001) Enhancing Engineering Design and Analysis Interoperability—Part 1: Constrained Objects. 1st MIT Conf. Computational Fluid & Structural Mechanics, Boston.

<sup>8</sup> Some references are available online at [www.pslm.gatech.edu](http://www.pslm.gatech.edu) and <http://eislabs.gatech.edu/>.

<sup>9</sup> <http://www.pslm.gatech.edu/events/frontiers2006/>

<sup>10</sup> X = Models throughout the product lifecycle including design (DAI), manufacturing (MAI), and sustainment (SAI).

## Author Biographies

**Russell S. Peak** is Associate Director and Senior Researcher in the Product & Systems Lifecycle Management Center at the Georgia Institute of Technology ([www.pslm.gatech.edu](http://www.pslm.gatech.edu)) and R&D Director at InterCAX LLC.

He specializes in information technology and knowledge-based methods for modeling and simulation (M&S). His interests include product lifecycle management (PLM) frameworks based on open standards and knowledge representations that enable complex system interoperability. Applications include systems engineering and simulation-based design (SBD) in domains such as electronic packaging, airframe structural analysis, and systems-of-systems (SoS).

Dr. Peak originated the multi-representation architecture (MRA)—a collection of patterns for CAD-CAE interoperability—and composable objects (COBs)—a non-causal object-oriented knowledge representation. This work has provided a conceptual foundation for executable parametrics in OMG SysML and associated validation.

**Roger M. Burkhardt** develops information technology architecture for engineering and other areas at John Deere in Moline, Illinois. His work emphasizes the use of computer-based models to enable collaboration across diverse business and technical concerns. Previously at Deere, he developed decision-support systems for factory design and production agriculture, simulation tools for multi-agent systems, and software frameworks for manufacturing planning. He has been one of the main developers of the structural aspects of OMG SysML.

**Sanford A. Friedenthal** is a Principal System Engineer for Lockheed Martin Corporation. His experience includes the system life cycle from conceptual design through development and production on a broad range of systems. He has been manager for systems engineering responsible for ensuring systems engineering processes are implemented on the programs.

He has been a lead developer of advanced systems engineering processes and methods including the Lockheed Martin Integrated Enterprise Process, the Software Productivity Consortium's Integrated Systems and Software Engineering Process, and the Object-Oriented Systems Engineering Method (OOSEM). Mr. Friedenthal also led the Industry Standards effort through the Object Management Group and INCOSE to develop OMG SysML™ that was adopted by the OMG in May, 2006.

**Miyako W. Wilson** is a Research Engineer in the Manufacturing Research Center at the Georgia Institute of Technology. She received her Masters degree from Georgia Tech in the School of Mechanical Engineering. Her research interests include computer-aided engineering and manufacturing and information technology. She is a lead software developer for related engineering applications, including composable object (COB) capabilities in *XaiTools* and associated interfaces to SysML tools.

**Manas Bajaj** is a PhD candidate in the School of Mechanical Engineering at the Georgia Institute of Technology. His research interests are in the realm of knowledge representation, open standards and simulation-based design in the context of product / system lifecycle management. He has authored several publications, including those that won the Robert E. Fulton best paper award at ASME CIE 2005 and the session best paper award at Semicon West 2003. He was an intern with the IBM Extreme Blue program, a co-op at Rockwell Collins, and is a part time co-op at InterCAX LLC.

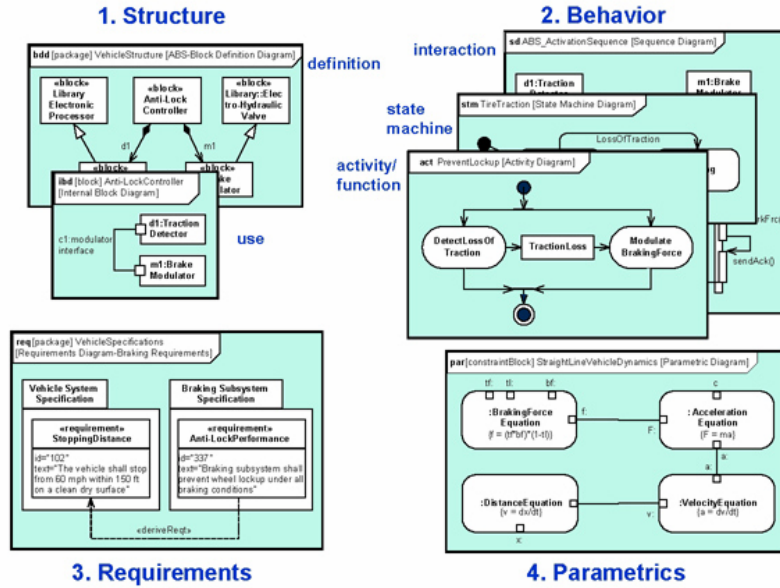
Mr. Bajaj has been actively involved in Georgia Tech contributions to the OMG SysML specification. He also serves as a technical team member in the PDES Inc. consortium.

**Injoong Kim** received his B.S. and M.S. in Mechanical Engineering at Yonsei University, Seoul. He is pursuing the Ph.D. degree in the School of Mechanical Engineering at the Georgia Institute of Technology. Prior to joining to Georgia Tech, he researched in the area of computer-aided engineering at Samsung Heavy Industries and Orominfo for five years. His research interests include knowledge-based engineering systems, computer-aided simulation for reliability, and engineering information systems.

# Appendix A

## A.1 More about SysML

Figure 11 overviews the four main capability areas of SysML, and Figure 12 provides several of the value types used in these Part 1 and Part 2 papers.



Note that the Package and Use Case diagrams are not shown in this example, but are respectively part of the structure and behavior pillars

Figure 11: The four pillars of SysML [OMG, 2007a].

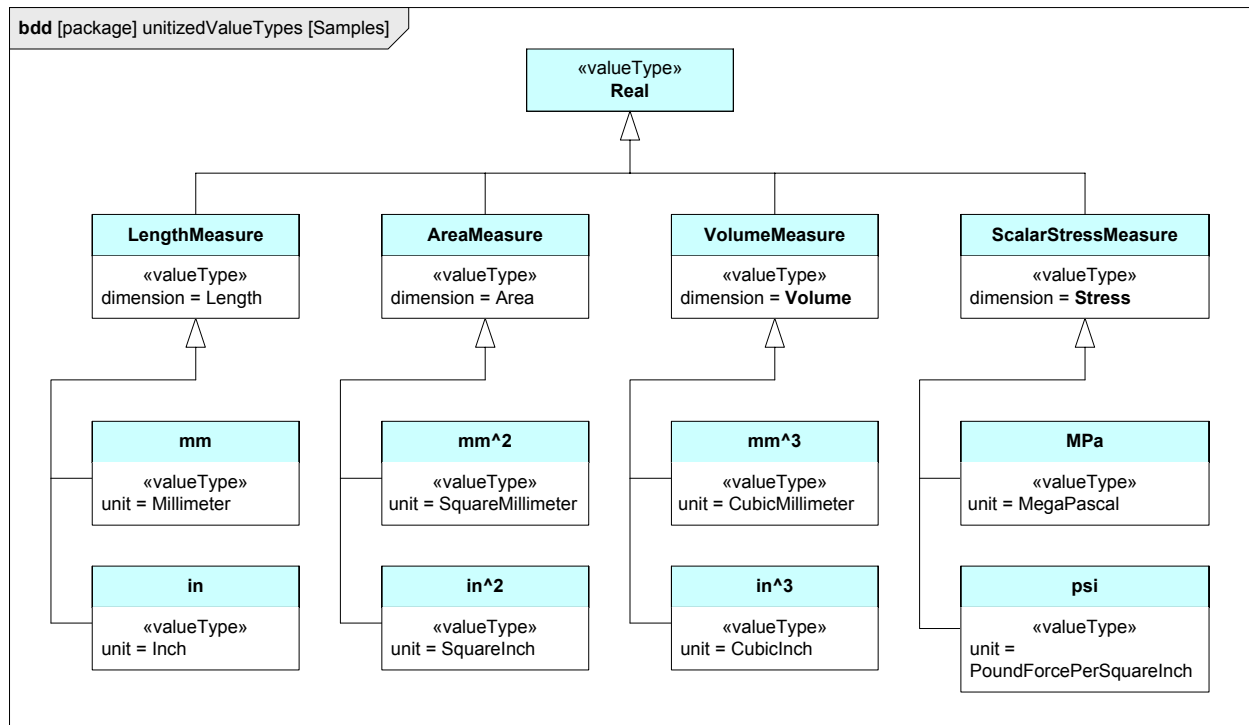


Figure 12: Sample value types used in SysML examples in Parts 1 and 2—based on spec App. C.4 [OMG, 2007b].

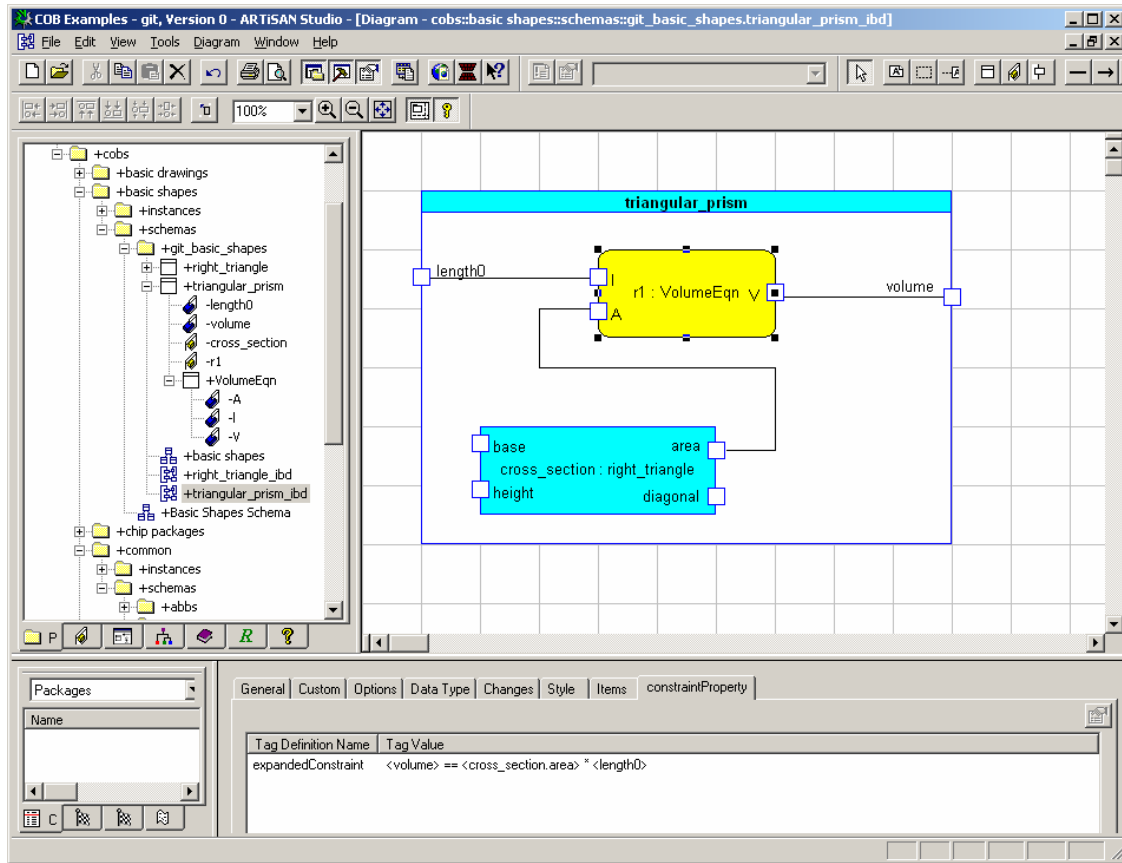


Figure 13: Sample parametrics implementation in a SysML tool (based on an early SysML spec draft).

## A.2 More about Composable Object (COB) Technology

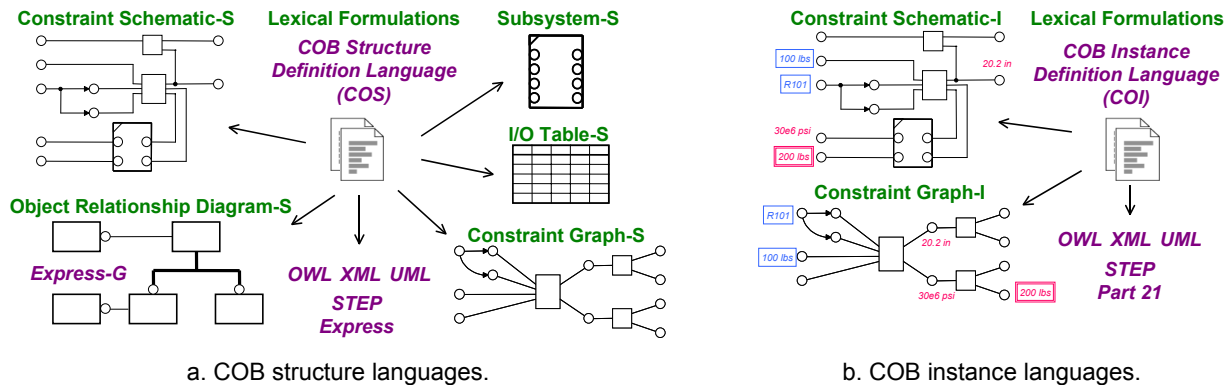


Figure 14: Classical lexical and graphical formulations<sup>11</sup> of the COB representation.

Work began at GIT in 1992 on the technology that eventually led to what we today call composable objects (COBs). See [Peak, 2000] for further background including early engineering analysis applications to electronics. COBs were originally known as *constrained objects* because of their constraint graph basis. Since 2005 they have been called *composable*<sup>12</sup> *objects* to better reflect their interactive model construction nature, which vendor support for SysML parametrics is beginning to make practical.

<sup>11</sup> XML versions of COS and COI are available as of Fall, 2005 (see Figure 8). STEP EXPRESS (ISO 10303-11) is an object-flavored information modeling standard geared towards the life cycle design and engineering aspects of a product [<http://www.nist.gov/sc4/>]. See [Peak et al. 2004] regarding relationships among STEP, XML, and UML.

<sup>12</sup> The word *composable* here is inspired by the composable simulation work that Prof. Chris Paredis and colleagues initiated at CMU [Paredis et al. 2001].

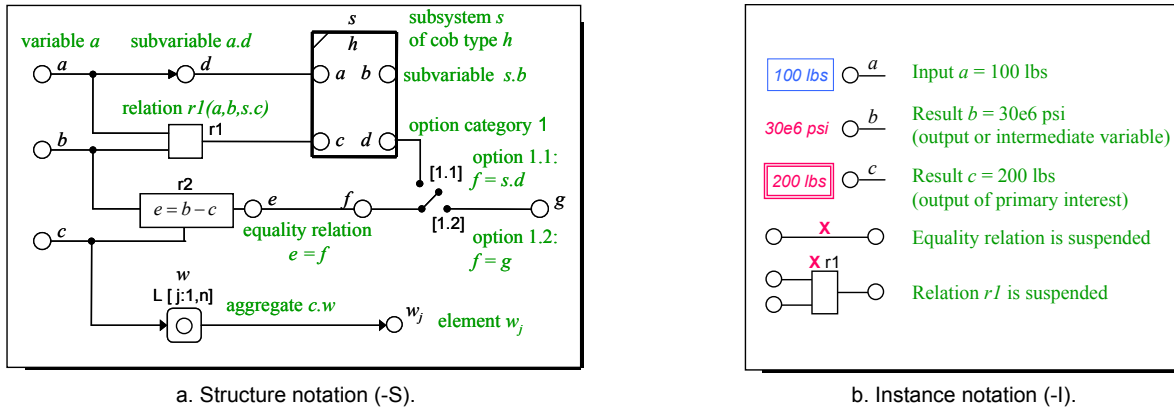


Figure 15: Basic COB constraint schematic notation.

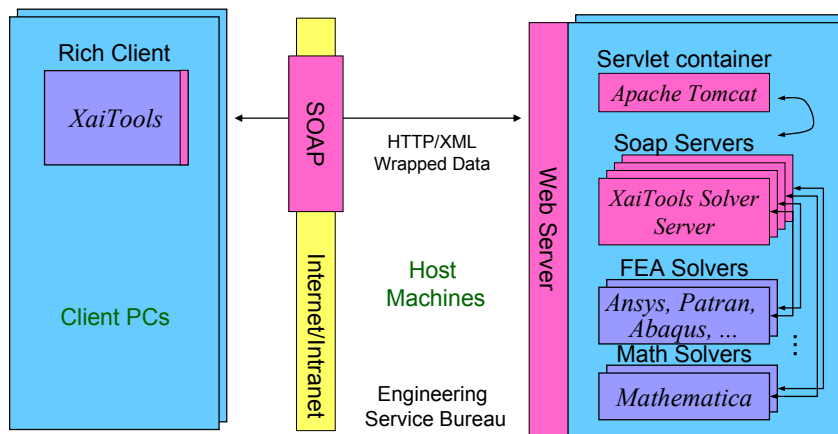


Figure 16: CAE solver access via *XaiTools* engineering web services.

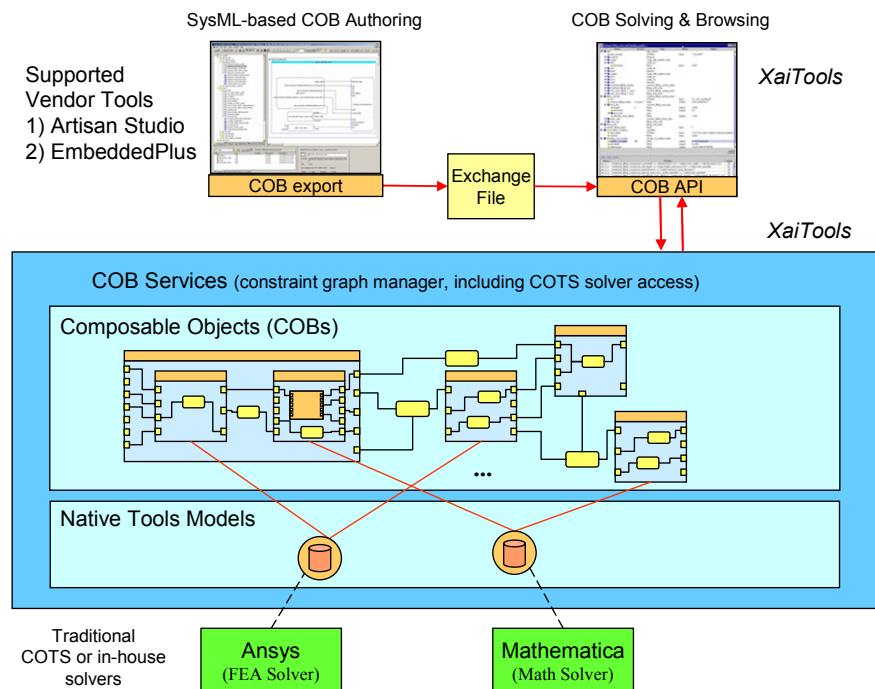


Figure 17: Prototype COB-based architecture for executable SysML parametrics.