The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs)

COB Requirements & Objectives (v1.0)

October 31, 2005

Submitted to:
National Aeronautics and Space Administration (NASA)
Goddard Space Flight Center (GSFC)
http://www.gsfc.nasa.gov/
Greenbelt, Maryland USA

Technical Contact:
Stephen C. Waterbury
stephen.c.waterbury@nasa.gov • +1-301-286-7557

Submitted by:
Product & Systems Lifecycle Management (PSLM) Center
http://www.pslm.gatech.edu/
Georgia Institute of Technology (GIT)
Atlanta, Georgia USA

Principal Investigator:
Russell S. Peak, PhD
russell.peak@marc.gatech.edu • +1-404-894-7572

Co-Investigator
Christiaan J. J. Paredis, PhD
chris.paredis@me.gatech.edu • +1-404-894-5613

Lead Researcher:
Diego R. Tamburini, PhD
diego.tamburini@marc.gatech.edu • +1-404-385-7159

Project Web Site:
http://eislab.gatech.edu/projects/nasa-ngcobs/

Table of Contents

List of Figures

List of Tables

Abstract

This document formulates a vision for advanced collaborative engineering environments (CEEs) to aid in the design, simulation and configuration management of complex engineering systems. Based on inputs from experienced Systems Engineers and technologists from various industries and government agencies, it identifies the current major challenges and pain points of Collaborative Engineering. Each of these challenges and pain points are mapped into desired capabilities of an envisioned CEE System that will address them.

Next, we present a CEE methodology that embodies these capabilities. We overview work done to date by GIT on the composable object (COB) knowledge representation as a basis for next-generation CEE systems. This methodology leverages the multi-representation architecture (MRA) for simulation templates, the user-oriented SysML standard for system modeling, and standards like STEP AP233 (ISO 10303-233) for enhanced interoperability. Finally, we present COB representation requirements in the context of this CEE methodology. In this current project and subsequent phases we are striving to fulfill these requirements as we develop next-generation COB capabilities.

Document History

Version	Date	Changes
1.0	2005-10-31	First published version.

Project Team

Primary Team:

NASA GSFC

- Steve Waterbury

GIT

- PI: Russell Peak
- Co-PI: Chris Paredis
- Lead Researcher: Diego Tamburini
- Technical Team:
  - Manas Bajaj
  - Injoong Kim
  - Miyako Wilson

Advisory Team:

- Jim U'Ren (NASA JPL - AP233 Team Lead)
- Sandy Friedenthal (Lockheed Martin - SysML Team Lead)

Acknowledgements

We thank the following individuals for their valuable feedback on this document and/or its components: Alan Moore (Artisan Software), Roger Burkhart (Deere & Co.), Peter Benson (ECCMA), Doug Clark (GARD Associates), Laurent Balmelli (IBM), Mike Dickerson and Dirk Zwemer (InterCAX), Jay Brusse (Muniz Engineering), Jim Andary, Dave Everett, Harry Frisch, and Gary Mosier (NASA GSFC), Clark Briggs, Steve Cornford, Steve Jenkins, Bob Oberto, Steve Prusha, Farrokh Shoar, Georg Siebes, Joe Skipper, Luke Voss, Steve Wall, and Becky Wheeler (NASA/JPL), Joe Collins (Naval Research Laboratory), Matt Aronoff, Conrad Bock, Kevin Brady, Peter Denno, Steven Fenves, Simon Frechette, Josh Lubell, John Messina, and Ram Sriram (NIST), Mark Austin and Natasha Shmunis (University of Maryland).

This feedback came via various interactions, including during the following project meetings (contact the authors if you would like access to these materials):

- NASA GSFC :
  - June 1 & 3, 2005 (Greenbelt, MD)
- NASA JPL:
  - July 5-7, 2005 (Pasadena, CA)
  - September 23, 2005 (Pasadena, CA)
  - October 18, 2005 (Pasadena, CA)
- NIST:
  - June 2, 2005 (Gaithersburg, MD)

We would also like to acknowledge the following groups whose members also provided input for this document:

- INCOSE Model-Driven System Design (MDSD) Working Group
- OMG Systems Engineering Domain-Specific Interest Group (SE DSIG)
- PDES, Inc.

Nomenclature

- ABB: Analysis Building Block
- API: Application Programming Interface
- APM: Analyzable Product Model
- CBAM: Context-Based Analysis Model
- COB: Composable Object
- CEE: Collaborative Engineering Environment
- CMS: COB Management System
- COTS: Commercial Off-The-Shelf Software
- DBMS: Database Management System
- GIT: Georgia Institute of Technology
- GUI: Graphical User Interface
- MRA: Multi-Representation Architecture
- ngCOB: Next-Generation COB
- PSLM: Product & Systems Lifecycle Management
- PLM: Product Lifecycle Management
- SE: Systems Engineer
- SMM: Solution Method Model
- SoS: Systems-of-Systems

Disclaimer

This document may identify commercial product names and materials to describe certain procedures or to provide concrete examples (i.e., to help clarify abstract concepts via specific instances). In no case does product or material identification imply recommendation or endorsement by the authors or their organizations, nor does it imply that such items are necessarily the best available for the purpose. Company, product, or service names may be included that are trademarks or service marks of others.

1 Document Purpose

The purpose of this document is to formulate a vision for advanced collaborative engineering environments to aid in the design, simulation and configuration management of complex engineering systems. This document is one of the deliverables for Phase 1 of the project "*Constrained Object Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs)*" (overviewed in the next section) as outlined in the project proposal [see EIS Lab 2005].

With this goal of vision formulation in mind, the authors held a series of meetings and interviews with experienced practicing SEs and technologists from NASA, JPL, LMCO and NIST to identify the current major challenges and pain points of collaborative engineering. The input from these meetings is captured in Section 3.1 ("CEE Challenges and Needs"). This list of challenges, as well as the authors' combined experience conducting research in the area of design-analysis integration and engineering knowledge modeling representation, was used to compile a list of desired capabilities that a CEE System should have to address these pain points (Section 3.2 – "Desired Capabilities of a CEE System"). Section 4.1 presents an envisioned CEE system and its components that embody these capabilities. Section 4.2 ("Composable Objects/Multi-Representation Architecture (COB/MRA) ") overviews the work done to date by GIT on the COB Representation and the requirements it shall meet to enable next-generation CEE systems. For all the above lists, this document focuses more on capturing items relevant to the COB methodology rather than completeness.

Figure 1 illustrates the mapping sequence followed by this document from challenges to requirements; starting from the CEE Challenges, mapping each challenge to CEE Desired Capabilities to address them, and in turn mapping these capabilities to the subsystems of GIT's COB Platform that implement them. Also shown is how we map the desired capabilities to requirements specific to the COB Representation (the conceptual foundation of the COB Platform, described in Section 4.2.1).

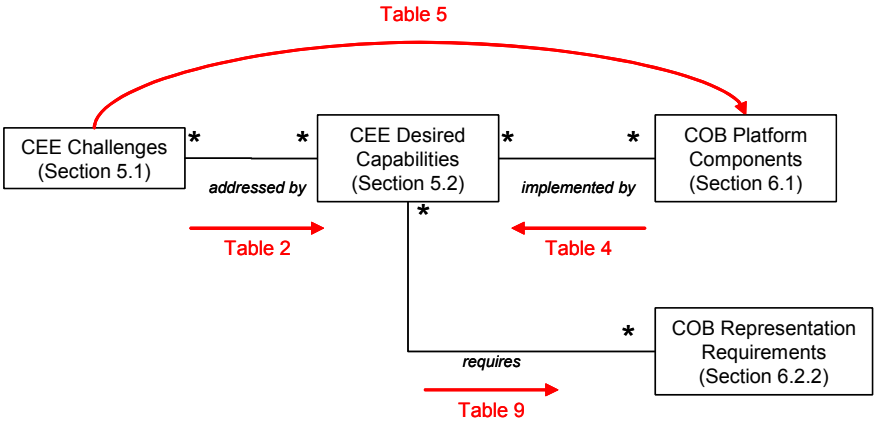


Figure 1: Mappings between CEE Challenges, Capabilities and Requirements

The vision outlined in this document is at a rather high-level. The idea is that, once this document captures what we believe to be a reasonably complete list of requirements, we will prioritize them and determine which are in scope for our next phases of development. We will then add enough detail to the requirements in scope to make them testable and to drive the development of prototypes (and, ultimately, production-quality systems based on commercialization of these techniques).

2 Project Background

This document is the first deliverable of Phase 1 of the NASA Goddard-funded project "The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs)". This project is the result of GIT's response to NASA's Engineering for Complex Systems Collaborative Engineering Environment 2003 Call for Proposals. In this work, we are exploring the use of our COB methodology to support collaborative analysis and decision-making in space mission designs.

During this phase, we are identifying the needs and pain points of collaborative engineering that should be taken into consideration for the design of collaborative engineering environments, and documenting the software system requirements to address them. For more details please see the project web pages [EIS Lab 2005].

3 Problem Overview

3.1 CEE Challenges and Needs

This section describes the main challenges, pain points and needs of Collaborative Engineering[1] that should be taken into account for the development of software-based systems and tools to support advanced collaborative engineering environments. This list was compiled by the authors based on feedback obtained during a series of meetings and interviews with experienced practicing SEs at NASA and JPL as well as from the authors' own experience conducting research and executing projects in the area of design-analysis integration and modeling of composable objects (or COBs - see Section 4.2 for an introduction). These challenges will provide the context for the presentation of the desired capabilities of a CEE system in the next section.

- ***CHAL-001: Complex team dynamics:*** Collaborative Engineering brings together a team of specialists from multiple disciplines with the common objective of designing a system that meets the mission's goals in the most cost-effective way. They need to share, negotiate and exchange information with the other members of the team, and these interactions must be carefully orchestrated to keep the impact of any changes they make in their subsystems to the rest of the system under control.
- ***CHAL-002: Lack of end-user tools for collaborative Systems Engineering:*** although there is an abundance of mature and well-established software tools to aid in the design, analysis and simulation of individual components in the system (which is the focus of *Design* Engineering), there is a lack of end-user software tools for managing the complex interconnections of the components being designed to their supersystem and subsystems (the focus of Systems and Collaborative Engineering). The collaboration tools currently available focus on the real-time collaboration and information sharing amongst members of the team during a design session, providing functionality such as web-based real-time messaging, remote visualization of CAD models, and mark-up. While these tools add sharing capabilities to traditional PLM systems, they do not directly address the interconnection of information between models and disciplines.
- ***CHAL-003: Multidisciplinary:*** Collaborative Engineering is multidisciplinary by nature. Individual team members represent different disciplines, and/or individual subsystems such as Propulsion, Thermal, Structures, Communications, etc. Each discipline generally uses different software tools, information models, graphical nomenclatures, etc., which of course makes communication with others difficult at best. It is not practical to solve this problem by forcing everyone to use the same tools and information models. Instead, a mechanism should be provided to share information in various formulations (including graphical views) that are understood by everybody, while at the same time accommodating the individual disciplines' tools and models.
- ***CHAL-004: Focus shift over the project's lifecycle:*** the goals, focus, team dynamics and information generated during a Collaborative Engineering project vary widely depending on the phase of the project's life cycle. In general, earlier phases have these characteristics:

  1) They focus on negotiating and establishing the parameters shared amongst subsystems, performing multi-disciplinary system trades, proposing alternative concepts, and performing trade studies to determine at least one design that is feasible (and ideally to identify several designs and their gradients at points that are considered to be the most cost-effective in a generalized sense),

[1] Since *Systems Engineering* is, by nature, a complex collaborative engineering activity (and, conversely, since *Collaborative Engineering* is normally aimed at the design of systems), when we refer to the challenges, pain points and needs of Collaborative Engineering we are also referring to those of Systems Engineering.

2) They tend to be short in duration.

3) They are highly dynamic and interactive.

4) They resort to quick, back-of-the-envelope calculations.

As the project evolves the focus shifts into adding detail to subsystems and finally to the individual components and performing more detailed analysis and calculations, which normally requires the use of specialized software tools. This shift in focus notwithstanding, teams should be able to build on top of the results from the previous phases, recreating as little information as possible. Also, at the end of each stage, the information is normally packaged in some way (normally a written report) for review and for approval to move into the next phase (for example, at NASA a PDR – Product Design Review – is held at the end of Phase B to determine if they can proceed onto Phase C).

To illustrate the shift in focus of Systems Engineering throughout the life of a project, Table 1 below lists the various phases of a NASA project (as described in [NASA 1995]), and the main goal of each phase.

Table 1: Project Life Cycle for Major NASA Systems

Phase	Name	Goal
Pre-A	Advanced Studies	Produce alternatives and ideas from which new programs/projects can be selected. Prepare program/project proposals for consideration.
A	Preliminary Analysis	Determine the feasibility and desirability of a suggested major system
B	Definition	Define the project in enough detail to establish an initial baseline capable of meeting mission needs. Baseline the "design-to" specifications.
C	Design	Complete the detailed design of the system. Baseline the "build-to" specifications.
D	Development	Build the subsystems and integrate them to create the system. Baseline the "as-built" specifications.
E	Operations	Operate the system and dispose it properly

- ***CHAL-005: Complex information interconnections:*** in any sizeable system, the information connections between subsystems and their parameters may quickly grow into a complex graph of relations that is hard to visualize and manage. In addition, these relations may not have a predefined input/output direction (in other words, they are generally *non-causal*). As a result, the effect that a change (of a parameter value, mathematical model, assumption, constraint, etc.) has in other parts of the system becomes harder to predict, while at the same time the sensitivity of the overall system (the degree to which changes in one part of the system impact other parts of the system) increases. In other words, the system becomes less "resilient" to changes. Consequently, the team may incur costly rework when they realize later in the design process that a seemingly simple change made in one system has a costly effect in another system that was not caught in time. System-level what-if scenarios also become increasingly difficult to perform.
- ***CHAL-006: Multiple system views:*** a system may have multiple views depending on the criteria used to organize it. For example, the subsystems in a system may be organized depending on their function, position in the system, discipline, and geographical location. A CEE system must be able

to manage multiple simultaneous views of the same system, allowing users to switch to any of these views at any time to view or edit and maintaining consistency among them.

- ***CHAL-007: Information-intensive trade studies:*** at various points of the project lifecycle, SEs may need to perform trade studies to select amongst multiple plausible alternatives that meet the goals and objectives of the system (effectiveness, cost, schedule, and risk – both quantifiable and otherwise). Trade studies vary widely in complexity and detail depending on the context and the phase of the project life in which they are performed, but generally speaking they involve the following steps (for a detailed description of the Trade Study Process, see Section 5.1 of [NASA 1995]):

    1. Define the *system's goals and objectives*, and identify the *constraints* it must meet. As the project evolves, these will take the form of quantifiable performance requirements that a system must meet.
    2. Define the *outcome variables* (measures of system effectiveness, system performance or technical attributes, and system cost) and the *measurement methods* that are going to be used to compute each. This step explicitly identifies the variables that are important in meeting the system's goals and objectives. It also identifies the mathematical models that are required to evaluate them.
    3. Define the *selection rule*, which defines how the outcome variables are going to be used to make a selection of the preferred alternative.
    4. Define *plausible alternatives*: alternatives that can potentially meet the goals and objectives of the system, in other words, that are in the "solution space".
    5. *Collect data* on each plausible alternative and evaluate the outcome variables using the measurement methods.
    6. *Compute estimates* of the outcome variables.
    7. Make a *tentative selection*.
    8. Perform a "*reality check*" (scrutinize the results, measurement methods, conformance to goals, objectives and constraints, etc.)
    9. If the tentative selection holds up to the reality check, *proceed* with it for further resolution or implementation. The estimates of the outcome variables obtained in step 6 serve as inputs for the next resolution iteration.

    From the steps above we can see that trade studies are information- and computation-intensive processes that require careful management of the system variables, the interconnections amongst them, and the methods used to compute them. Performing these studies manually – although possible – is very time-consuming, costly and error prone.

- ***CHAL-008: Uncertainty:*** when evaluating values of outcome variables in a system, some of the inputs, models or measurement methods being used may not be well known (e.g., environmental impact) or are inherently random (e.g., manufacturing processes). As a result, the values of the outcome variables will have varying degrees of uncertainty. Often this uncertainty is taken into account only implicitly—an analyst may include a measure of the uncertainty based on experience (e.g., the stress analysis has an error of +/- 10%). If the impact of the uncertainty is considered to be significant, then the SE may perform a sensitivity analysis or even a Monte Carlo simulation to determine the uncertainty in the output more accurately.

    Although much of the Systems Engineering practice is still based on the use of safety factors to account for uncertainty, increasingly the use of probabilistic representations is gaining acceptance. With the rapidly decreasing cost of computing resources, probabilistic design methods such as Design For Six Sigma (DFSS) have become feasible and more reliable alternatives to deterministic methods [Koch, 2002]. Yet, few engineering environments currently include tools for uncertainty management. To allow uncertainty to be managed effectively, tools should include explicit representations of uncertainty, support computational tools for propagating uncertainty from requirements to performance assessments, and provide methods for decision making based on uncertain information. For a more detailed overview of how uncertainty impacts the Systems Engineering process, see [Aughenbaugh, 2004].

- ***CHAL-009: Capturing decisions and decision rationale:*** SEs should be able to re-trace the trail of decisions made about the system as it evolved through its project life cycle. This may be needed for a variety of purposes such as documentation, auditing, project reviews, learning, supporting other lifecycle processes (e.g., ranging from long-term sustainment and operation to knowledge reuse when initiating future systems) and so on. For example, they may need to show why a given design alternative was chosen against others during a trade study, or why a particular mathematical model was used instead of another. The decisions that are made are generally entered into a configuration management system as changes to (or elaborations of) the system baseline. However, this type of change tracking (and associated rationale capture) does not necessarily occur during the early stages of the design, when the baseline has not yet been established and therefore when a configuration management system is not being used yet. During the early stages of the design, any decision capture mechanism faces the additional challenge of having to be as unobtrusive as possible, particularly during the formulation and evaluation of multiple trade studies when agility is particularly critical.
- ***CHAL-010: Capturing assumptions and applicability of models:*** for a given calculation there might be multiple mathematical models that are applicable - depending on the level of detail required or the computation cost we are willing to incur. SEs should be able to access the assumptions under which a given model is applicable to make an informed decision as to which model to utilize. Similarly, the assumptions and other relevant information that underlie constraints should be also captured, so that it is possible to estimate the effect of relaxing them if needed.
- ***CHAL-011: Variety of software tools and information standards:*** as the project evolves, the level of detail required demands the use of both generalized and specialized software tools (COTS or home-grown) for the design, analysis and simulation of the various components of the system. The number of such tools may grow quite large, and sending and retrieving information to and from these tools (and performing the necessary data conversions to do so) may become an arduous and time-consuming task and drastically reduce the agility of the Systems Engineering process. Some existing information exchange standards (such as ISO STEP) attempt to alleviate this problem. However, sometimes there are several overlapping standards in the picture, and therefore we still need to deal with translations and transformation of information.
- ***CHAL-012: Multiple dimensions of versioning control:*** there are at least four independent dimensions of versioning control involved in the design of a system:
    - *Versioning Control of Design Models:* individual components that make up the system are constantly versioned to capture modifications made to them. For example, the dimensions of a flap link may be modified to reduce localized stress, resulting in a new version of the CAD model of the flap link. This versioning is normally managed with traditional COTS PLM tools, but this operation also needs to be coordinated with the overall CEE.
    - *Versioning Control of System Models:* models of the top-level system and major subsystems are themselves also constantly revised, for example, as subsystems, output variables and connections are added or modified. Each of these modifications effectively results in a new version of these system models.
    - *Versioning Control of Computation Models / Simulation Templates:* when selecting a computation model (say, to solve for system variables or to assist in a trade study), SEs often resort to models that have been used (and proven successful) in previous missions. They may use these computation models as-is (if the usage conditions and assumptions are similar), or modify them to accommodate different usage conditions. Some models are reused often enough that they warrant capturing in the form of modular, reusable simulation template libraries (see ***CHAL-015***). Overall, SEs and domain engineers need to be able to retrieve specific versions of models from previous designs and from simulation template libraries, and to modify and store new versions of models.
    - *Versioning Control of Analysis/Simulation Sets:* multiple analyses or simulations may be performed on a given version of the design models and the system model. For example, a

simulation may be performed using different sets of inputs, selecting different computation models, or selecting different system configurations. SEs need to be able to store the entire set used for an analysis/simulation run to be able to retrieve it and reproduce the results at any time.

Also related to versioning control is the need to capture "snapshots" of the overall system. At key points of the project lifecycle, a snapshot of all the information is used to hold a review and assess the readiness to move to the next phase. This snapshot will be composed of the artifacts listed above at their current version at the time the snapshot. Hence, SEs need a mechanism to create such a snapshot and "tag" it for future retrieval (similar to the way software developers can tag a group of independently versioned files in CVS).

- ***CHAL-013: Traceability of requirements:*** at each level of system resolution, SEs need to relate the overall system's requirements to specific (and - particularly as the project detail grows – *quantifiable*) goals and constraints on subsystems and their parameters. Conversely, SEs should be able to trace the goals and constraints of a particular subsystem up to the original requirements they are realizing. For example, a high level requirement for a given system may be: "shall operate at temperatures of up to 100 F". This requirement may translate into several requirements for a leaf-level component like a printed circuit board (PCB), one of which may be "the deformation at the midpoint of the PCB shall be less than 0.001 inch at 100 F". A SE should be able to determine at any time if an arbitrarily deep subsystem (in our example, the PCB) satisfies the system requirements mapped to it. Current requirements management tools allow users to capture requirements in a central database and even linking these requirements to objects in other tools (such as design tools), but do not currently provide a generalized mechanism for automatically (and continuously) validating these requirements (directly from the design tools, or based on diverse analysis/simulation results).
- ***CHAL-014: Workflow:*** a model of the system provides a global definition of how the subsystems in a system are composed together and how their parameters are related and constrained. It does not, however, explicitly indicate *the sequence of steps* to populate (determine the values of the parameters of) the system; that is, which parameters (and in what order) should be provided as inputs, which selections should be made (and when), and which relations will be solved (and in which direction). For complex systems, the number of possible combinations of all these may grow unwieldily large. In some cases, however, experience has identified a specific sequence of steps (or *workflow*) to successfully populate a particular system. Test scenarios could also be captured as workflows (as in "given a specific sequence of inputs, this are the outputs we expect"). For these cases, it would be valuable for a CEE system to provide a capability for capturing and executing these workflows.
- ***CHAL-015: Modularity and reusability:*** large systems are rarely modeled completely from scratch. Some components are similar from one project to another, and hence there is always some level of reusability that occurs. This situation exists for many types of models (e.g., requirements patterns, simulation methods, geometric design features, off-the-shelf hardware, and so on). It occurs more frequently with fundamental, general-purpose building blocks (for example, a fundamental analytical concept such as a 1-D Linear Elastic Material Model), but it may also occur with larger, pre-assembled purpose-specific subsystems. For example one study on airframe structural analysis estimates there are several hundred reusable generic analytical concepts (for structural analysis in general) and on the order of 10,000 airframe-specific structural analysis templates [Peak, 2003]. CEE systems should provide the ability to define modular, reusable system components (namely, subsystems and relations) and store them in libraries, so than can be used as building blocks for building other systems.

3.2 Desired Capabilities of a CEE System

This section describes the desired capabilities of an envisioned CEE System to address the CEE Challenges and Needs listed in the previous section. These capabilities are realized by the use cases represented in the UML Use-Case diagram in Figure 14 (page 17). Table 2 at the end of the section (page 18) summarizes the

capabilities that address each challenge listed, and Table 3 (page 19) summarizes the use cases that realize these capabilities.

A CEE System should provide:

- ***CAP-001: End-user tools for collaborative Systems Engineering:*** to allow a team of SEs to collaboratively create, edit and navigate system models. As illustrated in Figure 2, this includes defining the subsystems within the system, system public and private parameters, parameter constraints, relations between and within subsystems, relations between parameters and connections with external authoring or analysis tools. It should also include the ability to formally delegate the detailed modeling of a subsystem to another member of the team - defining the boundary conditions (required parameters, constraints on parameters, inputs, etc.) within which the other member shall model the subsystem- and the ability to view the details of the systems designed by other members of the team (subject to appropriate permissions). These tools should be preferably graphical, but should also support alternative ways to define the system for varying user skill levels. For example, lexical editors for advanced users, graphical tools like SysML parametric diagram editors for intermediate users (ala electrical schematics), and model-based diagrams and domain-specific user interfaces for novice users.

  To make such a model-based approach efficient, the tools should include reference model libraries of COTS components and previous design solutions. This will allow SEs to develop system models more efficiently by re-using the knowledge stored in composable reference models. (A reference model is a structured container of information about a component or sub-system which includes models of the structure, the function and behavior of the component) [Paredis, 2001].

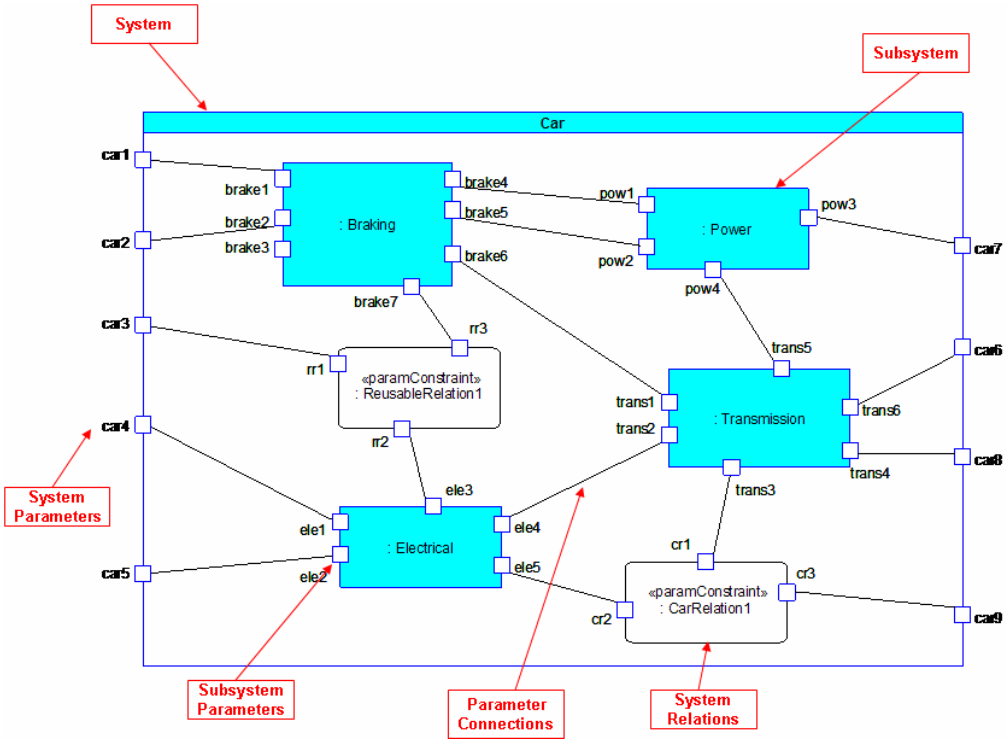


Figure 2: A System Model of a Car (using SysML notation [2])

- ***CAP-002: A common graphical notation for modeling and simulating systems:*** A common (preferably standard) way to graphically describe the interfaces between systems to facilitate inter-discipline communication and understanding of the inter-connections between systems, while at the

[2] The SysML notations used in this document roughly correspond to SysML draft v0.9 plus more recent updates and experimental variations. We intend to update these examples with the final official notation when SysML v1.0 becomes available.

same time allowing each discipline to use and maintain their own preferred graphical notations independently. Figure 2 above shows a representation of a system using SysML [SysML].

- ***CAP-003: Constructs for modeling complex system relations:*** to allow SEs to model the wide range of interconnections (or relations) that may occur in a system. Among the constructs this toolset should include are the following (Figure 3 illustrates each of these constructs[3]):
  - *Formula-based relations:* for defining relations between parameters that can be expressed as an equation (algebraic or otherwise, including equalities and inequalities)
  - *Equality relations:* a special case of formula-based relation (the simplest) for defining when the value of a parameter must be equal to the value of another parameter anywhere else (either in the same or in another system).
  - *Constraint relations:* a special type of formula-based relation defined on one parameter used to constraint its value.
  - *Aggregate relations:* for defining operations on parameters that are aggregates instead of single-valued (e.g., to obtain the average, minimum, maximum value).
  - *Buffered relations:* for defining relations among parameters that allow one or more parameters to vary within a certain range before affecting associated parameters. The goal of this type of interconnection is to reduce the "brittleness" (i.e., the susceptibility of a system to be impacted by changes in another system) of a system.
  - *Selectors:* similar to switches in an electrical circuit; used for defining alternative interconnections whose selection is based on certain predefined conditions (e.g., logical conditions such as "if the value of "a" in subsystem A is greater than 10, then connect it to "b" in subsystem B, otherwise connect it to "c" in subsystem C") or user choice at run time. These interconnections are also known as higher order relations.

    This construct is particularly useful for capturing models of *multiple levels of fidelity*. For example, a model of a flap link may define two cross sections of the flap link: a simplified and a detailed one. During a preliminary stress analysis, analyst may want to choose the simplified cross section to perform the analysis, but later in a more detailed analysis he may want to choose the detailed cross section to obtain more accurate results.
  - *Breakers:* similar to traditional breakers in an electrical circuit; connections that can be automatically or manually "deactivated" when certain conditions occur. Breakers may be treated as a special type of selector.
  - *Black-box relations:* for defining relations whose details, solution algorithms and execution method are encapsulated in an external block of executable code or an external software tool accessible via an API. Here, the relation should only define what the parameters involved are and any connection parameters needed to "talk" to the external software component. Examples of this type of relations are finite-element analyses, external software components (SOAP web services, Java APIs, COM components), and humans (largely based in heuristics and experience and that have not been captured in computable form - what we have come to call "Ask Bob" relations).
  - *Unidirectional relations:* Relations should be generally considered as multidirectional (that is, *non-causal*—the input/output direction is not specified in advance) to allow for multiple execution routes of the same system graph and enable model reusability for different scenarios. However, there are cases where relations are inherently causal (i.e., they cannot run in multiple directions) so the system should also provide a construct to support this special case that we denote as unidirectional.

  In addition, these constructs should also provide support for capturing uncertainty of a relation and/or probabilistic distributions of the input variables to allow the calculation of uncertainty of the values calculated.

[3] For simplicity, relations in this figure are shown exclusively between two systems (A and B) and involving a minimal number of parameters, but they could also occur *inside* one system, or *among more than two systems*, and involve *any number of parameters*.

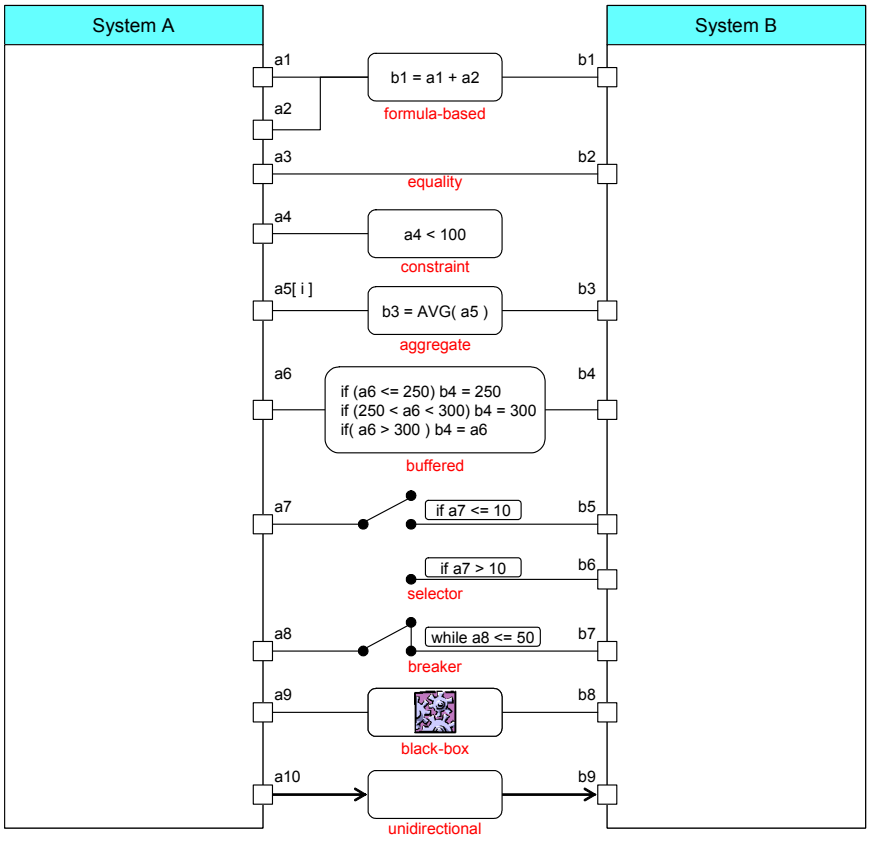


Figure 3: System Relations Types

- ***CAP-004: End-user tools to visualize the complex interconnections in a system:*** to allow users to quickly assess the impact of any changes they make (to design models, system models, relations, etc.) to the rest of the system and take corrective measures if needed. For example, Figure 4 illustrates a change in a design model (a change inside the Braking subsystem). Here, a user should be able to quickly assess the impact of changing something in the design model (say, the diameter of the sleeve of the flap link shown). In this example, the value of the parameter "car6" changes to 12.5 as a result of this Braking subsystem change. The user should also be able to see the effect of this change anywhere in the system. This ability should not be restricted to changes in *design models*, but should ideally consider changes in any part of the system (system models, relations, constraints, etc.).

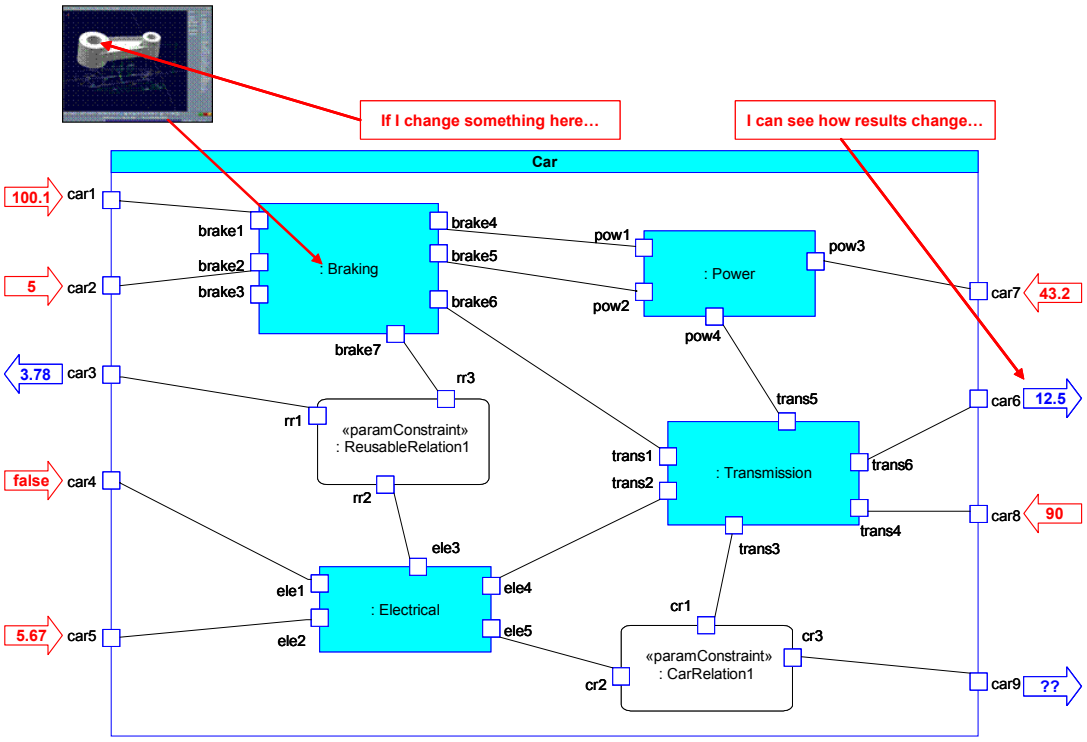


Figure 4: Impact of a Change in a System Model

Another desired feature is to be able to visualize the margin or "slack" of a system model (Figure 5); that is, what can be changed (and to what extent) in a subsystem without impacting the rest of the system.

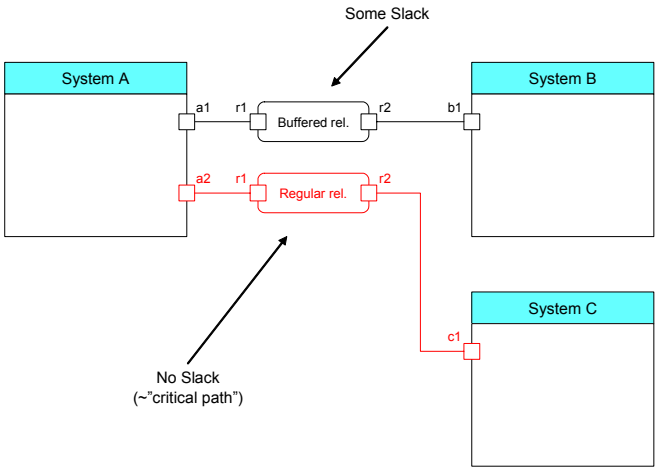


Figure 5: Slack in a System

- ***CAP-005: An approach to manage consistency of shared parameters throughout the project lifecycle:*** early in the project, subsystem leads may agree that their subsystems share a parameter. They may also agree on constraints on its value. For example, as illustrated in Figure 6(a), the leads of subsystems A and B may agree to share the weight of Subsystem A (indicated by a line connecting parameters weight in Subsystem A and weight-a in Subsystem B), and that its value must not exceed 100 lb. Once this agreement is made, each subsystem lead may proceed to add detail to his or her subsystem, possibly redefining the way a shared parameter is calculated. For example, continuing with our example of Figure 6, in (b), the weight of Subsystem A is now calculated using a relation (r1) with input

from a solid model and other subsystem data. The risk now is that someone may make a change to the solid model (say, a design engineer using a CAD tool) that will increase the weight of Subsystem A above 100 lb, therefore "breaking" the agreement between subsystems A and B and potentially requiring costly rework of Subsystem B. Complicating the matter, as the system becomes more complex the number of parameters affecting the weight may increase, while at the same time becoming farther removed from the actual shared parameter (in other words, the number of intermediate relations between the affecting parameters and the shared parameter becomes larger). To control this situation, the CEE system must provide a way to constantly check that these "agreements" among subsystem are not being broken whenever a change is made, and if they must be broken, aid in the determination of what needs to be done as a result.

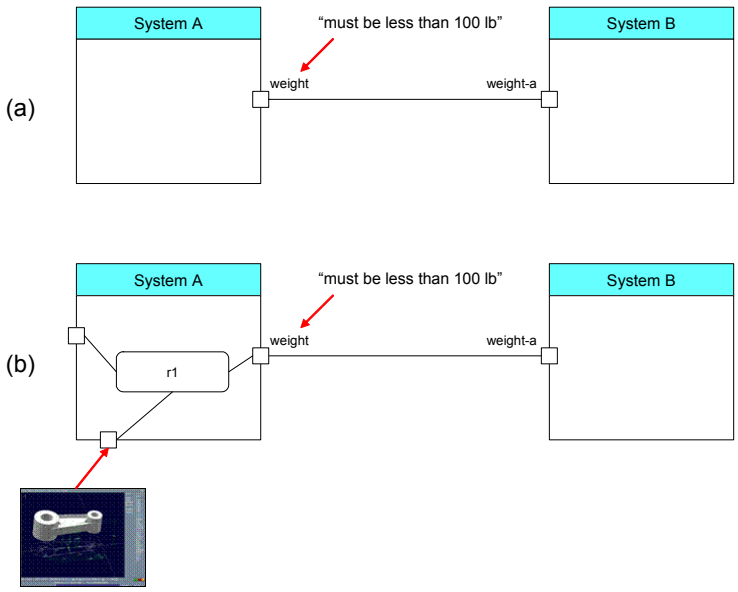


Figure 6: Maintaining Consistency of Shared Parameters

CAP-006: A versioning and configuration control mechanism (and related end-user tools): to enable independent fine-grained versioning of design models, system models, computation models, and analysis/simulation sets, a well as the shared parameters and relations among such models. Figure 7 illustrates versioning of *system models*, where the model of the Car system is versioned independently from the contained systems (Braking, Power, Electrical and Transmission) as indicated by the version numbers in parenthesis. Figure 8 illustrates the versioning of a *design model* (the CAD model of a flap link in the lower left corner, labeled "Design Model V3.7"), a *simulation set* (labeled "Simulation Set #123"), and a *computation model* (the relation "ABS Relation1" labeled "Computation Model V1.4"). The figure also illustrates a specific set of inputs and outputs (arrows coming into and out of the car system, respectively) for that specific combination of item versions. The mechanism for versioning simulation sets should allow users to capture everything that was selected to run the simulation, so that the simulation can be retrieved at any time and re-run to obtain comparable outputs. This includes the versions of the system model and design models, the input values, and any run-time selections available (for example, where several analysis models with different levels of fidelity are available to solve for the same parameters).

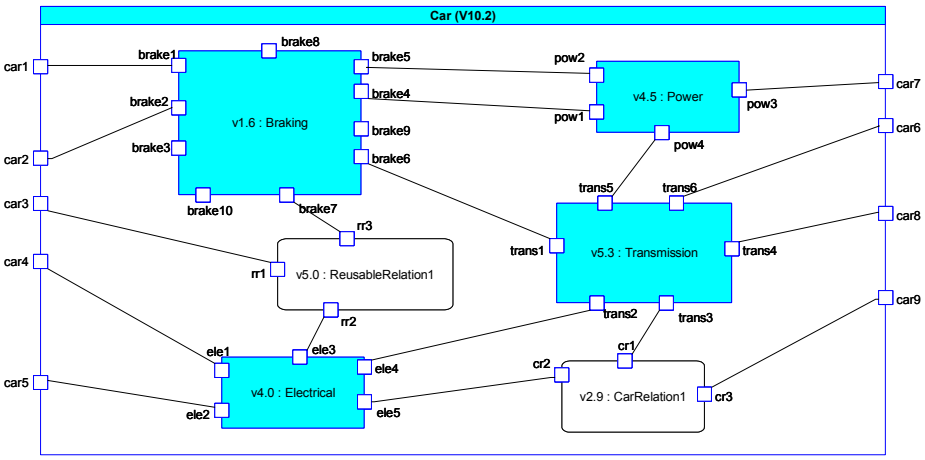


Figure 7: Version Control of a System Model

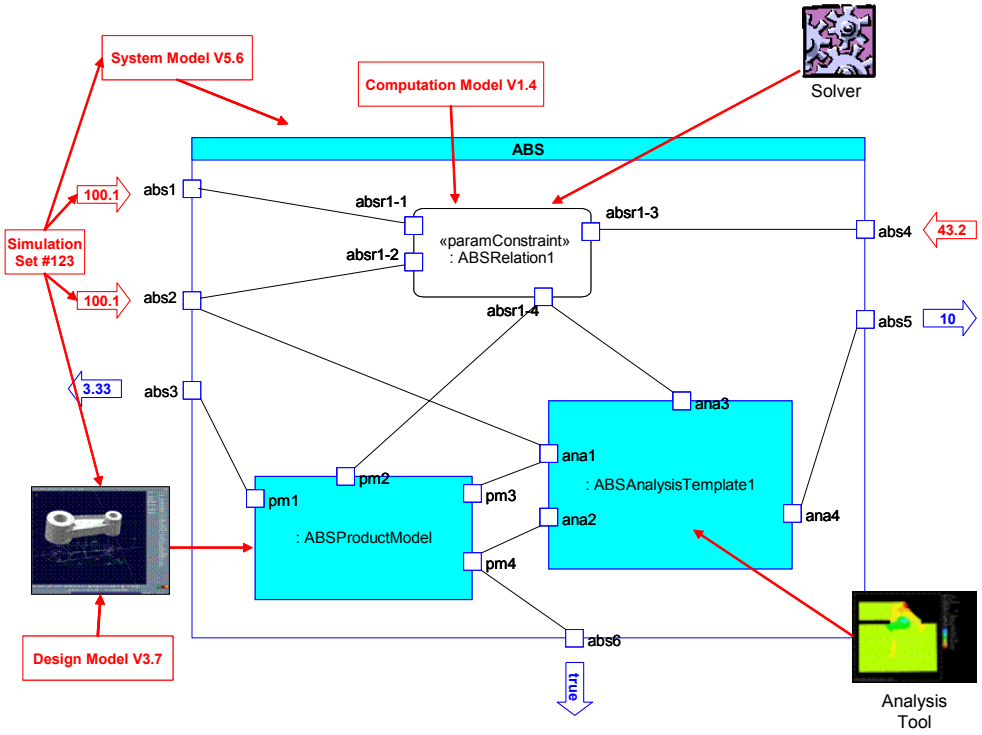


Figure 8: Versioning various aspects of a System Model

The versioning and configuration control mechanism should also provide "tagging" functionality (similar to the one provided by software versioning control systems such as CVS) to tag an arbitrary set of configuration-controlled artifacts for later retrieval. For example, at the end of Phase B in a NASA project all the system artifacts could be tagged as "PDR Version" (the version submitted to Preliminary Design Review"). The artifacts themselves may have different, independent versions (much like individual files tagged with the same tag may have different versions in CVS).

The versioning and configuration control mechanism should also provide the appropriate check in/check out logic to ensure consistency when a system is revised. To illustrate this, let's consider the simple example in Figure 9: two subsystems (System A.1 and System A.2) are part of a larger system (System A). Assume that at the beginning they are all at version 1.0. If someone creates a new version of System A.1 (version 2.0) and wants this version to be used in System A, then the owner of System A needs to check out System A and revise it to use version 2.0 of System A.1. If there are any

inconsistencies introduced by the new version of System A.1, they need to be resolved at this point (revising any of the systems to accommodate the changes). In our example, the change made to System A.1 does not introduce any inconsistency (only a new parameter - a12 - was added that will be used at the System A level later) and therefore a revision of System A to reflect the use of version 2 of System A.1 is sufficient (that is, no changes are needed in System A.2). The idea is that we want to prevent the owner of System A.1 from simply "pushing" the new version and potentially breaking the consistency of System A. In general, in order to use a new version of a system that has changes in functionality, the containing systems (and potentially the connected systems too) need to be checked out and revised as well.

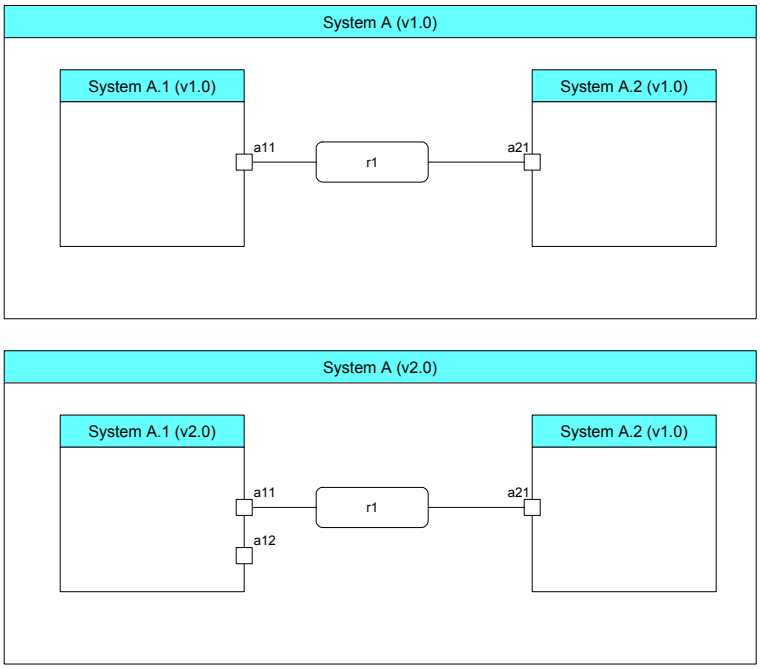


Figure 9: Versioning and Consistency

- ***CAP-007: The ability to create libraries of reusable components:*** to allow users to utilize and add these reusable components when building other systems. This functionality is particularly useful to capture commonly used analysis templates. Figure 10 illustrates such an application, where the 1D Linear Elastic Model system is used twice in the Extensional Rod and Torsional Rod systems (which in turn can be used in larger systems – not shown). This capability could be expanded beyond reusable analysis templates up to entire reusable systems that can be used as building blocks when composing larger systems. There also needs to be a way to capture the assumptions and constraints under which these reusable systems can be used (see CAP-008). It is important to include enough information about the component and its internal functioning to allow someone considering reusing it to determine that the component indeed works for the application it is intended. Only when people "trust" and understand a component will they be likely to reuse it in their models.

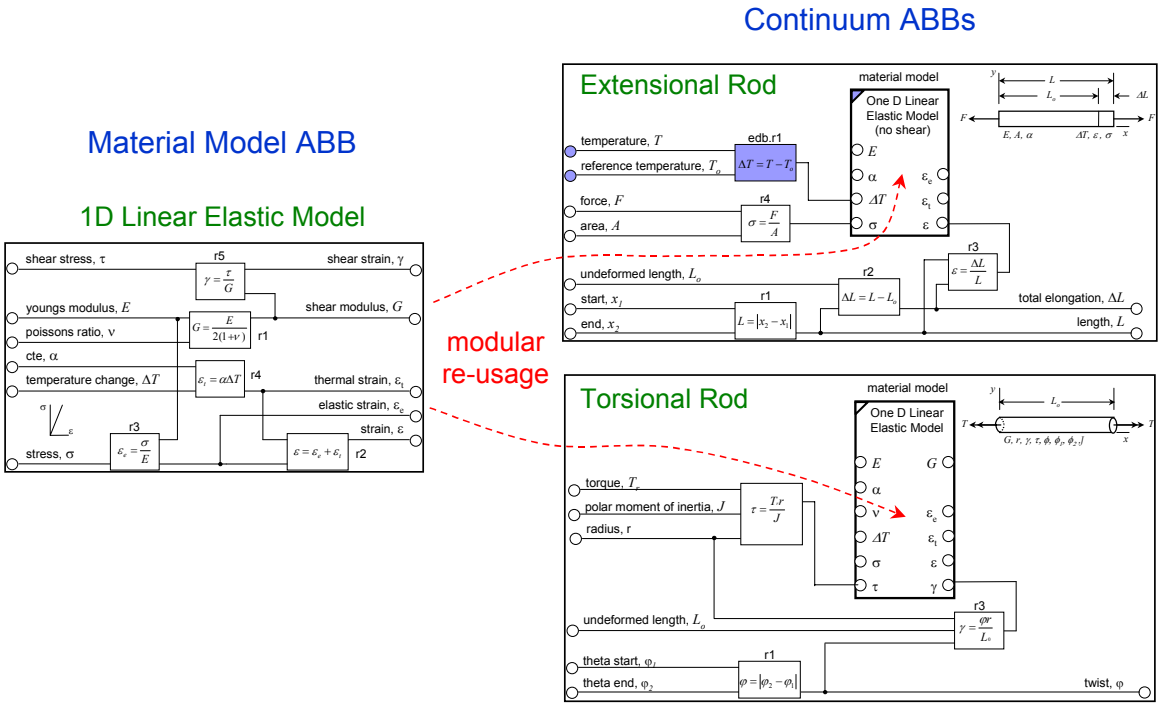


Figure 10: Analysis Building Blocks (ABBs) as Reusable System Components

- ***CAP-008: Ability to capture the assumptions, rationale and limitations of a model:*** to allow users to determine under which conditions it is appropriate to use the model (this is particularly important for low-fidelity models) and for enabling the automatic selection of valid models or reusable subsystems. For example, a given relation may only be valid for a certain range of values of one of its inputs. Another example is a subsystem (for example, one that represents a material model) that is best used for preliminary design (because is fast but approximate) versus another one that is best used for detailed design (because is computationally expensive, but accurate).

- ***CAP-009: Simulation orchestration:*** to allow users to run simulations (and view the results) of the behavior of a system (or any of its subsystems in isolation) and perform what-if scenarios under arbitrary input conditions. The connection to and execution of any underlying analysis or solver tools required to obtain output values should be transparent to the user running the simulation. The idea is that although domain and tool experts will still be required to *set up* a simulation, anyone in the team should be able to *execute* the simulation. Figure 11 illustrates a simulation set at the Car system level, where inputs are shown as incoming (red) arrows and outputs as outgoing (blue) arrows. Figure 12 illustrates the same simulation running within a lower-level subsystem (ABS) and how external modeling and solving tools are involved ("Design Tool" is used to retrieve some of the "pm" parameters in the "ABSProductModel" block, "Analysis Tool" is used to solve for some of the "ana" parameters in the "ABSAnalysisTemplate1" block, and "Solver" is used for some of the "absr1" parameters in the "ABSRelation1" relation).

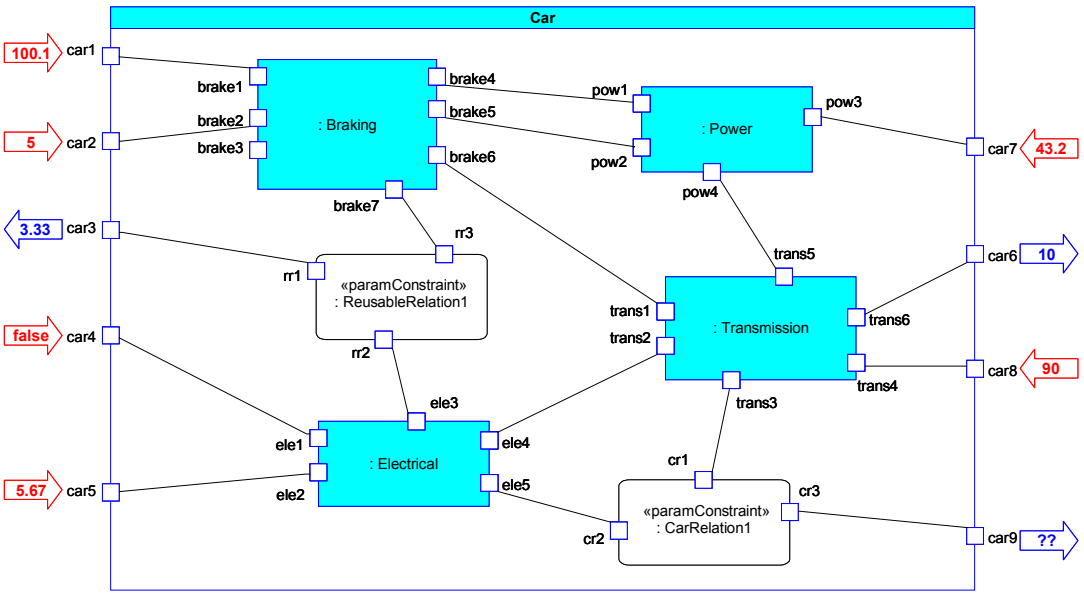


Figure 11: System Simulation

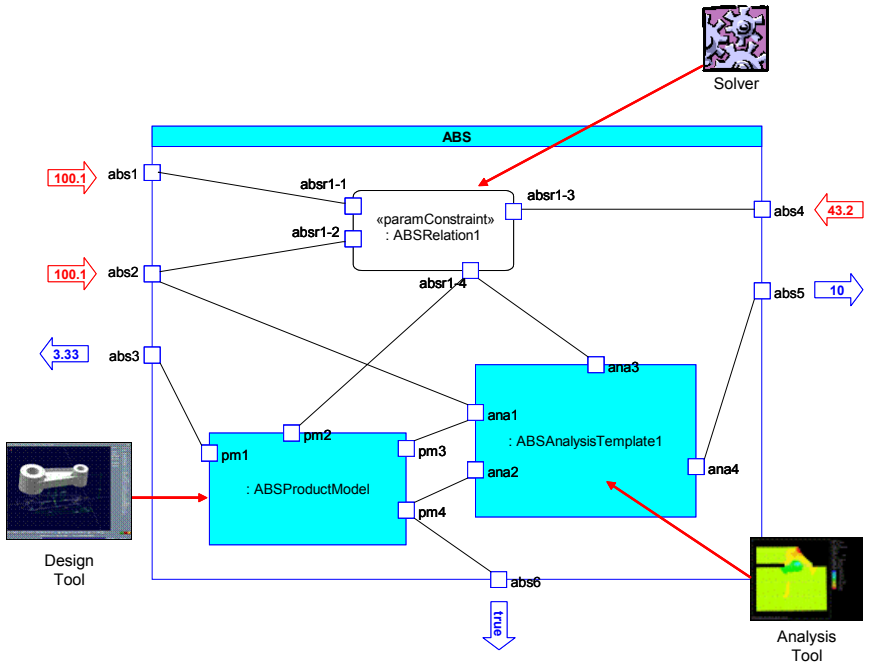


Figure 12: External Software Tools in a System Simulation

In addition, the system should also support simulation scenarios where some of the parameters have not yet been calculated (say, because they come from other subsystems that are still in work), or cannot be calculated until the system is assembled together. This is particularly important to allow independent unit-testing for systems that are being developed in parallel.

- ***CAP-010: Requirements allocation and traceability:*** to allow users to allocate system requirements (quantitative or otherwise) to the components of the system implementing those requirements. To that end, the system should provide users the ability to translate higher level requirements into more specific requirements and keep the traceability between them. The system should also allow users to graphically assess conformance to requirements as they are selecting design alternatives or making

changes to their systems. Conversely, users should be able to assess what parts of the system will be affected if something does not perform according to specs (for example, if a parameter value is outside a specified valid range).

- ***CAP-011: The ability to create and execute workflows:*** to allow users to specify the sequence of steps that should be followed to populate the parameters of interest in a system. As illustrated in Figure 13, the idea is that users would "run" the model of a system through a workflow, which would walk the user through the sequence of inputs and decisions that need to be made (based on intermediate results, for example) in order to populate these parameters. As the model advances through the workflow, it becomes more and more defined, until all the parameters of interest are populated at the end of the workflow. The CEE system should provide the capability for defining, storing, and executing workflows.

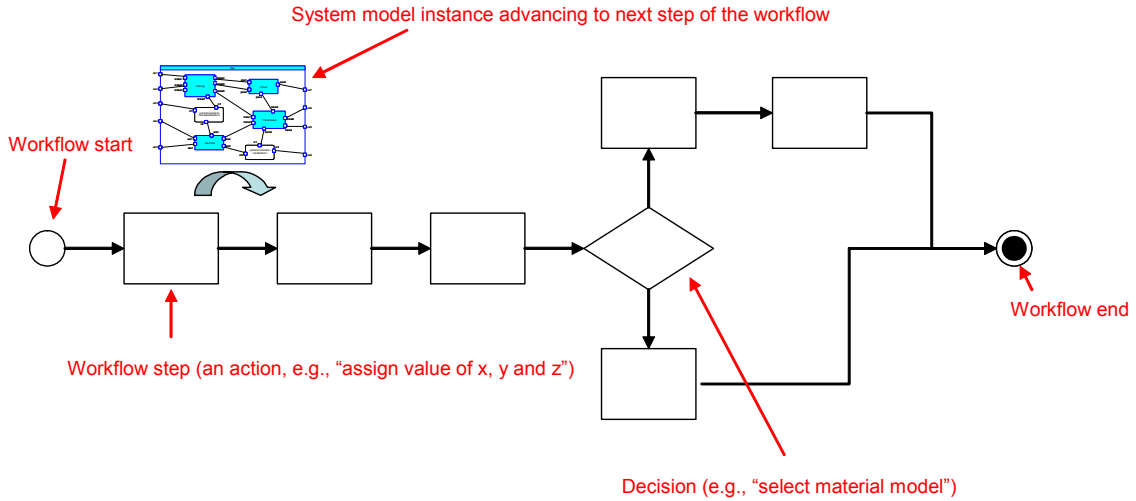


Figure 13: Workflow

- ***CAP-012: Ability to support trade studies:*** the system should provide the basic constructs and user interface to support the trade-study process described in CHAL-007 (Information-intensive trade studies) to capture multiple plausible alternatives and to select the "best" one to meet the goals and objectives of the system. It should allow users to model the system goals, objectives and constraints, outcome variables, measurement methods (mathematical models, information queries to external tools, etc.) and selection rules (selection algorithm or workflow). The system should then aid the user in identifying plausible alternatives (alternatives that meet the goals, or that are in the "solution space") and selecting the best alternative based on the selection rules.

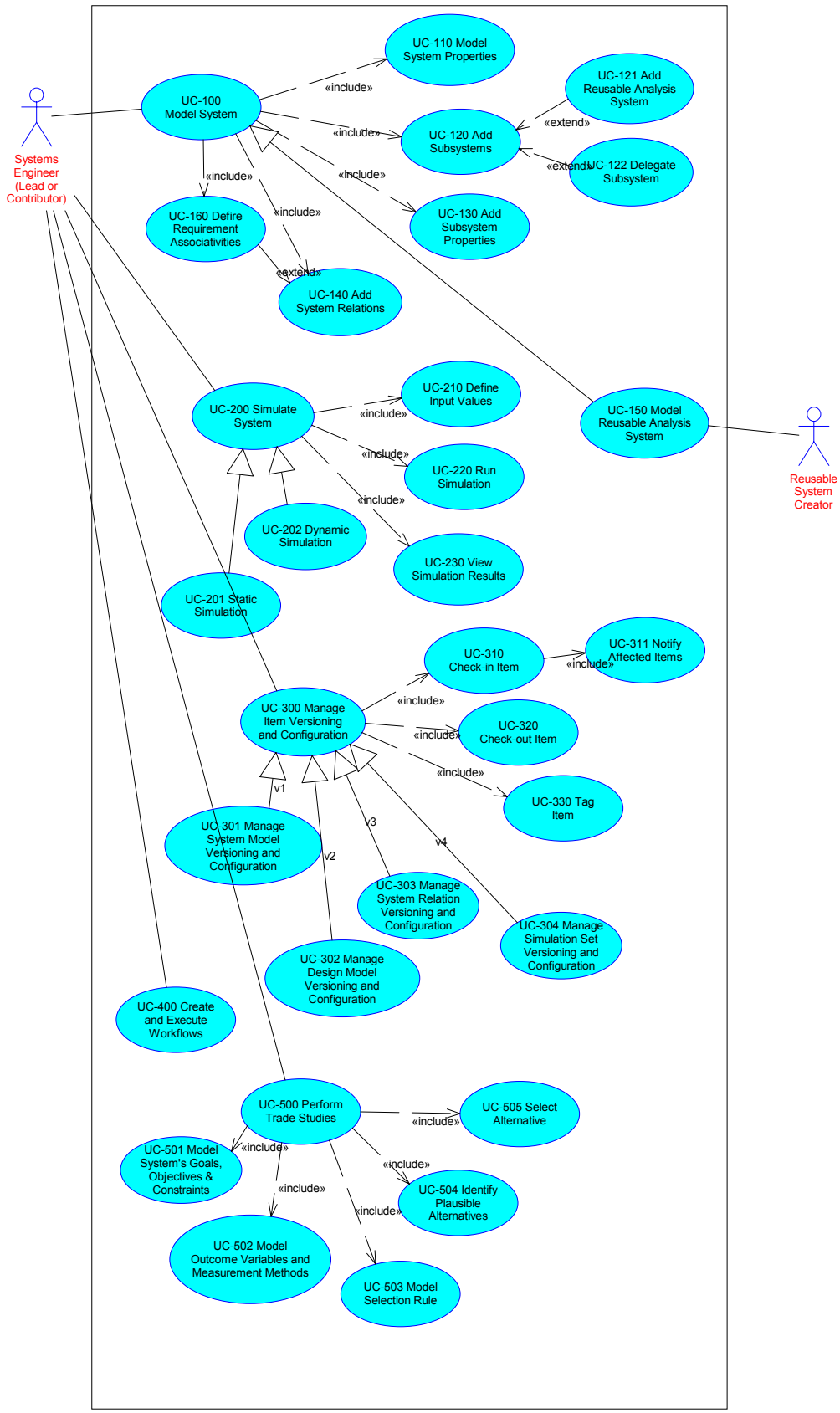


Figure 14: CEE System Use-Case Diagram

Table 2: CE Challenges and CEE System Desired Capabilities

Challenges	Capability(ies) Addressing Challenge
CHAL-001 (Complex team dynamics)	CAP-001 (End-user tools for collaborative Systems Engineering) CAP-002 (Common graphical notation for modeling and simulating systems) CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)
CHAL-002 (Lack of end-user tools for collaborative Systems Engineering)	CAP-001 (End-user tools for collaborative Systems Engineering)
CHAL-003 (Multidisciplinary)	CAP-001 (End-user tools for collaborative Systems Engineering) CAP-002 (Common graphical notation for modeling and simulating systems) CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)
CHAL-004 (Focus shift over the project's lifecycle)	CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle) CAP-006 (Versioning and configuration control mechanism)
CHAL-005 (Complex information interconnections)	CAP-003 (Constructs for modeling complex system relations) CAP-004 (End-user tools to visualize the complex interconnections in a system)
CHAL-006 (Multiple system Views)	CAP-003 (Constructs for modeling complex system relations) CAP-004 (End-user tools to visualize the complex interconnections in a system)
CHAL-007 (Information-intensive trade studies)	CAP-012 (Ability to support trade studies) CAP-003 (Constructs for modeling complex system relations) CAP-007 (Ability to create libraries of reusable components) CAP-009 (Simulation orchestration) CAP-011 (Ability to create and execute workflows)
CHAL-008 (Uncertainty)	CAP-003 (Constructs for modeling complex system relations)
CHAL-009 (Capturing Design Decisions and Rationale)	CAP-006 (Versioning and configuration control mechanism)
CHAL-010 (Capturing assumptions and applicability of models)	CAP-008 (Ability to capture the assumptions, rationale and limitations of a model)
CHAL-011 (Variety of software tools and information standards)	CAP-003 (Constructs for modeling complex system relations) CAP-009 (Simulation orchestration)
CHAL-012 (Multiple dimensions of versioning control)	CAP-006 (Versioning and configuration control mechanism)
CHAL-013 (Traceability of requirements)	CAP-010 (Requirements allocation and traceability)
CHAL-014 (Workflow)	CAP-011 (Ability to create and execute workflows)

Challenges	Capability(ies) Addressing Challenge
CHAL-015 (Modularity and Reusability)	CAP-007 (Ability to create libraries of reusable components)

Table 3: CEE Use Cases

Desired Capability	Use-Case(s) realizing Capability
CAP-001 (End-user tools for collaborative Systems Engineering)	UC-100 (Model System) (and related use cases)
CAP-002 (Common graphical notation for modeling and simulating systems)	UC-100 (Model System) (and related use cases)
CAP-003 (Constructs for modeling complex system relations)	UC-140 (Add System Relations) (and related use cases)
CAP-004 (End-user tools to visualize the complex interconnections in a system)	UC-140 (Add System Relations) (and related use cases)
CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)	UC-140 (Add System Relations) (and related use cases) UC-300 (Manage Item Model Versioning and Configuration) (and related use cases)
CAP-006 (Versioning and configuration control mechanism)	UC-300 (Manage Item Model Versioning and Configuration) (and related use cases)
CAP-007 (Ability to create libraries of reusable components)	UC-150 (Model Reusable Analysis System)
CAP-009 (Simulation orchestration)	UC-200 (Simulate System) (and related use cases)
CAP-010 (Requirements allocation and traceability)	UC-160 (Define Requirements Associativities)
CAP-011 (Workflow)	UC-400 (Create and Execute Workflows)
CAP-012 (Ability to support trade studies)	UC-500 (Perform Trade Study) (and related use cases)

3.3 CEE System Design Considerations

The following is a list of design considerations – beyond the ones imposed by the above functional capabilities - of CEEs that should be taken into account (and properly scoped) when designing a CEE system:

- Solving speed required
- Size and complexity of systems being developed
- Number of users
- Security requirements
- Scalability
- Maintainability
- Configurability

Given the same desired capabilities, different values for these considerations will generally result in different CEE systems (and possibly different CEE system architectures). For example, a CEE system for designing commercial products within a single small company will look quite different compared to a CEE system for designing classified systems among large primes and their subcontractors.

4 Envisioned COB/MRA-based CEE Methodology

4.1 Envisioned Next-Generation CEEs

4.1.1 Overview

Our envisioned CEE methodology to address the above challenges involves two new CEE System components (Figure 15): 1) a *Composable Object* (*COB) Management System (CMS)*, and 2) a software development platform, termed *COB Platform*, on top of which a CMS and other applications are built. This approach embodies the COB Representation that leverages constraint graph and object-oriented concepts to enable micro-level associativity among diverse models. This technology provides infrastructure services and algorithms (constraint meta-solving, tools-access orchestration, etc.) that bring together best-of-breed modeling, design, and analysis tools into a collaborative modeling and simulation (M&S) environment. It leverages SysML as a unifying graphical nomenclature for end users to compose federated simulations by connecting diverse model components via fine-grained relations.

The services this platform provides — embodied in CMS — can be coupled with existing enterprise CEE systems (like the NASA ESMD[4] Windchill-based CEE) or used to support next-generation localized environments (e.g., going beyond parameter-exchange servers like ICEMaker [Kevin 2003] or parametric modeling orchestrators like Phoenix Integration ModelCenter). Engineers employ SysML graphical modeling inside COB Platform-based applications to create new components and leverage reusable building blocks. In this way system and domain engineering teams collaboratively model, navigate, and simulate systems more intuitively and better visualize and manage their complex interdependencies.

Figure 15 shows the high-level architecture of the proposed COB Platform and CMS. The COB Platform is composed of a set of ***COB Services*** and a ***COB Software Development Toolkit (SDK)***, with the ***COB Representation*** shown along their side as their conceptual foundation. The CMS — developed using the COB SDK — is the deployable embodiment of the COB Platform, and comprises ***COB Server Components*** and ***COB Management Client Tools***. The CMS is itself a stand-alone COB-enabled application and provides the basic functionality and end-user tools for authoring and executing COB graphs.

Focusing on the COB Platform portion of the figure, the *COB Representation* is a constraint graph- and object-based knowledge representation developed by our team at the Georgia Institute of Technology (GIT). It provides the conceptual foundation for the COB Platform, as it defines the underlying information model and algorithms for implementing the COB SDK and COB Services. Section 4.2 provides an overview of GIT work to date on the COB Representation and the related design pattern, the Multi-Representation Architecture, that provides modular, reusable simulation template technology.

End-user tool developers use the *COB SDK* to build applications that leverage these services. The COB SDK provides a class library, UI components, development tools, proxies to access the COB Services, samples, documentation, and wrappers for some commonly-used external tools.

The *COB Services* provide the base functionality to help applications benefit from the use of constraint graphs as their underlying data representation model. These services include:

- *COB Graph Management Services:* for managing the lifecycle (creation, meta-solving, persistence, disposal) of COBs and COB graphs.
- *COB Versioning Control Services:* for controlling the versioning of COBs, including check in and check out, tagging, etc.
- *COB Persistence Services:* for storing and retrieving COBs to and from a persistent store (database).
- *COB Workflow Services:* for creating, executing and monitoring workflows that involve COBs undergoing state changes as they advance through their steps.
- *COTS Access Services:* for accessing external tools to retrieve or solve for data.

[4] ESMD = Exploration Systems Mission Directorate (http://exploration.nasa.gov/)

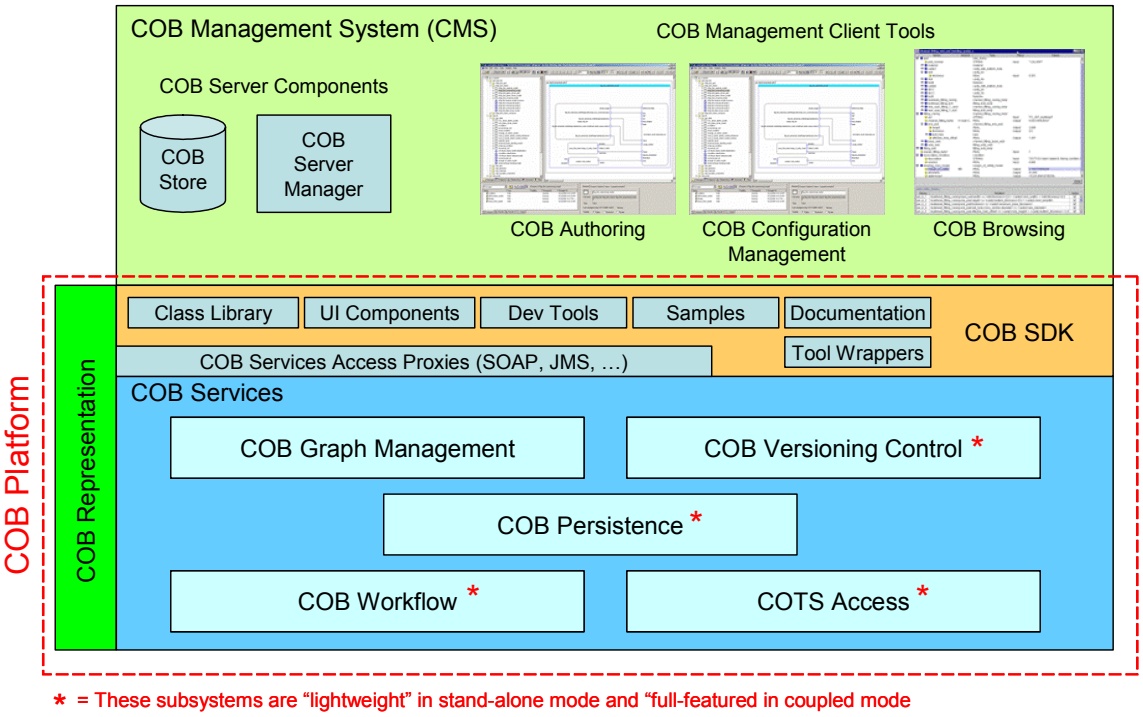


Figure 15: COB Platform and CMS High-Level Architecture

Some of these services are denoted in Figure 15 as "lightweight", meaning that they provide limited basic functionality in the case of the stand-alone deployment of CMS (CMS deployment is discussed later in this section). More complete functionality is achieved when the CMS is coupled with corresponding dedicated systems (such as a PLM systems, workflow engines, database management systems, or interoperability middleware).

Table 4 below lists the COB Platform components and the capabilities (from Section 3.2) enabled by each.

Table 4: CEE Capabilities Implemented by Each COB Platform Component

COB Platform Component	Enabled CEE Capability(ies)
COB SDK	Exposes the COB Services that implement the capabilities
CMS Management Client Tools	CAP-001 (End-user tools for collaborative Systems Engineering) (enables their development) CAP-004 (End-user tools to visualize the complex interconnections in a system)
COB Graph Management Services	CAP-001 (End-user tools for collaborative Systems Engineering) (enables their development) CAP-002 (Common graphical notation for modeling and simulating systems) (provides support for exchanging data with system modeling tools) CAP-003 (Constructs for modeling complex system relations) CAP-004 (End-user tools to visualize the complex interconnections in a system) (enables their development) CAP-005 (Approach to manage consistency of shared parameters throughout the project lifecycle) CAP-007 (Ability to create libraries of reusable components) CAP-008 (Ability to capture the assumptions, rationale and limitations of a model)

COB Platform Component	Enabled CEE Capability(ies)
	CAP-009 (Simulation orchestration) CAP-010: Requirements allocation and traceability
COB Versioning Control Services	CAP-006 (Versioning and configuration control mechanism) CAP-007 (Ability to create libraries of reusable components)
COB Persistence	CAP-006 (Versioning and configuration control mechanism) CAP-007 (Ability to create libraries of reusable components)
COB Workflow	CAP-011: The ability to create and execute workflows
COTS Access Services	CAP-009 (Simulation orchestration)

Table 2 is reproduced as Table 5, with a new column indicating which COB Platform component implements each capability. This table shows the path from the challenges listed in Section 3.1 in the first column, through the capabilities that address each challenge (Section 3.2) in the second column, to the COB Platform components that support these capabilities in the third column.

Table 5: COB Platform Components Implementing Each CEE Capability

Challenge	Capability(ies) Addressing Challenge	COB Platform Component Implementing Capability
CHAL-001 (Complex team dynamics)	CAP-001 (End-user tools for collaborative Systems Engineering)	CMS Management Client Tools COB Graph Management Services (enables their development)
	CAP-002 (Common graphical notation for modeling and simulating systems)	COB Graph Management Services
	CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)	COB Graph Management Services
CHAL-002 (Lack of end-user tools for collaborative Systems Engineering)	CAP-001 (End-user tools for collaborative Systems Engineering)	COB Graph Management Services (enables their development)
CHAL-003 (Multidisciplinary)	CAP-001 (End-user tools for collaborative Systems Engineering)	COB Graph Management Services (enables their development)
	CAP-002 (Common graphical notation for modeling and simulating systems)	COB Graph Management Services
	CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)	COB Graph Management Services
CHAL-004 (Focus shift over the project's lifecycle)	CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)	COB Graph Management Services
	CAP-006 (Versioning and configuration control mechanism)	COB Versioning Control Services COB Persistence
CHAL-005 (Complex information	CAP-003 (Constructs for modeling complex	COB Graph Management Services

Challenge	Capability(ies) Addressing Challenge	COB Platform Component Implementing Capability
interconnections)	system relations)	
	CAP-004 (End-user tools to visualize the complex interconnections in a system)	CMS Management Client Tools COB Graph Management Services
CHAL-006 (Multiple system Views)	CAP-003 (Constructs for modeling complex system relations)	COB Graph Management Services
	CAP-004 (End-user tools to visualize the complex interconnections in a system)	COB Graph Management Services
CHAL-007 (Information-intensive trade studies)	CAP-012 (Ability to support trade studies)	COB Graph Management Services
	CAP-003 (Constructs for modeling complex system relations)	COB Graph Management Services
	CAP-007 (Ability to create libraries of reusable components)	COB Graph Management Services COB Versioning Control Services COB Persistence
	CAP-009 (Simulation orchestration)	COB Graph Management Services COTS Access Services
	CAP-011 (Ability to create and execute workflows)	COB Workflow Services
CHAL-008 (Uncertainty)	CAP-003 (Constructs for modeling complex system relations)	COB Graph Management Services
CHAL-009 (Capturing Design Decisions and Rationale)	CAP-006 (Versioning and configuration control mechanism)	COB Versioning Control Services COB Persistence
CHAL-010 (Capturing assumptions and applicability of models)	CAP-008 (Ability to capture the assumptions, rationale and limitations of a model)	COB Graph Management Services COB Versioning Control Services
CHAL-011 (Variety of software tools and information standards)	CAP-003 (Constructs for modeling complex system relations)	COB Graph Management Services
	CAP-009 (Simulation orchestration)	COB Graph Management Services COTS Access Services
CHAL-012 (Multiple dimensions of versioning control)	CAP-006 (Versioning and configuration control mechanism)	COB Versioning Control Services COB Persistence
CHAL-013 (Traceability of requirements)	CAP-010 (Requirements allocation and traceability)	COB Graph Management Services
CHAL-014 (Workflow)	CAP-011 (Ability to create and execute workflows)	COB Workflow Services
CHAL-015 (Modularity and Reusability)	CAP-007 (Ability to create libraries of reusable components)	COB Graph Management Services COB Versioning Control Services COB Persistence

4.1.2 COB Management System (CMS)-based CEEs

As mentioned above, the envisioned CMS can be either deployed stand-alone or coupled with existing dedicated systems (such as PLM systems, workflow engines, database management systems, interoperability middleware, etc.). These two deployment options are illustrated in Figures 16 and 17, respectively. Figure 16 illustrates the stand-alone option, in which the server side provides the COB Services. The client side consists of a suite of tools ("COB Management Client Tools" in the figure) for managing the system and performing basic COB management tasks. Also shown in Figure 16 are COB-enabled end-user applications, which developers create using the COB API to access the CMS services on the server. The CMS Management Client Tools are essentially COB-enabled end-user applications too, with their only distinguishing characteristic being that they are delivered as part of the CMS.

While COB-enabled end-user applications would still have to implement the user interface (UI) and business logic specific to their application, they would leverage the graph solving, generic COB UI, information mapping, and tool access "plumbing" logic provided by the COB Services. The underlying constraint graph used by these applications does not need to be hard-coded in the application itself; instead, it can be stored and maintained independently and read by the application (and potentially by *several* COB-enabled applications concurrently) at run time.

Depending on the application, the underlying constraint graph structure may be relatively static and require little or no modification during the life of the application. Such applications leverage the graph to enable attribute value changes and input/output direction changes. These type of static COB graph structures may be created upfront (at application design time) using a SysML tool or simply by hand with a text editor (using the COB lexical form, for example). Other applications may want to provide users the ability to manipulate and modify the graph structure at run time; for these the COB SDK also provides UI Components than can be embedded in the application to display the constraint graph as a SysML diagram and enable interactive graphical manipulation.

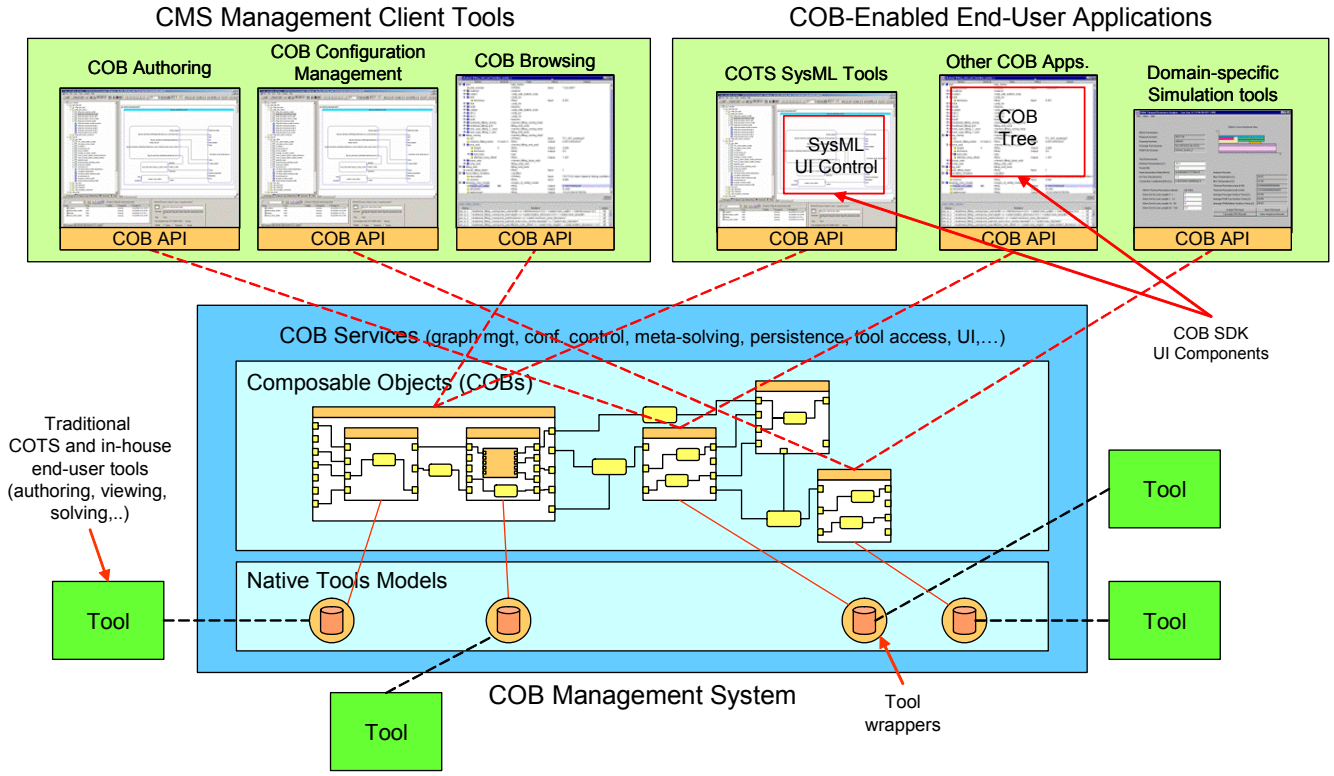


Figure 16: Stand-Alone COB Management System (CMS) and COB-Enabled Applications (e.g., in "next-generation ICEMaker" mode)

The vision is that, using COB-enabled applications, teams of engineers will be able to collaboratively and graphically (using SysML) model their systems and the complex interrelations between their parameters. They will assemble models by using a combination of reusable models from predefined libraries and new models they create from scratch. They will drag and drop graphically pluggable components — provided by tool vendors or third parties — that encapsulate the access to external tools and connect them to the rest of the system. Lastly, they will run simulations of these systems, and the underlying environment will orchestrate the execution of external software tools and the storage and retrieval of information behind the scenes.

The SysML diagram of the system is treated as a constraint graph representing the relations between the subsystems and their parameters, ranging from simple equality relations to complex algorithms implemented in external software tools such as finite element analysis (FEA) solvers. The direction in which these relations are executed is not assumed in advance; so a given relation may support multiple input/output scenarios. In other words, COBs provide a non-causal knowledge representation, which allows the same models to take on various causalities depending on appropriate I/O combinations at different points during the development of complex systems like space systems. Underneath, the COB Services manage the solving of the constraint graph and coordinate the access and execution of external tools to solve for the outputs. Freed from peripheral tasks such as coding, tool set up and execution coordination, and data re-entry, engineers can now focus on the design of the system itself, evaluate more alternatives, consider more what-if scenarios, and detect conflicts earlier — all this leading to better designs.

Figure 17 illustrates the second CMS deployment option, where existing dedicated systems (CEE/PLM systems, in this example) are coupled with a CMS. Here the CEE/PLM system and the CMS enhance each other's functionality; the CEE/PLM system provides industrial strength configuration management, access control, workflow and tool access services to the CMS, while the CMS provides graph management and meta-solving services to the CEE/PLM system.

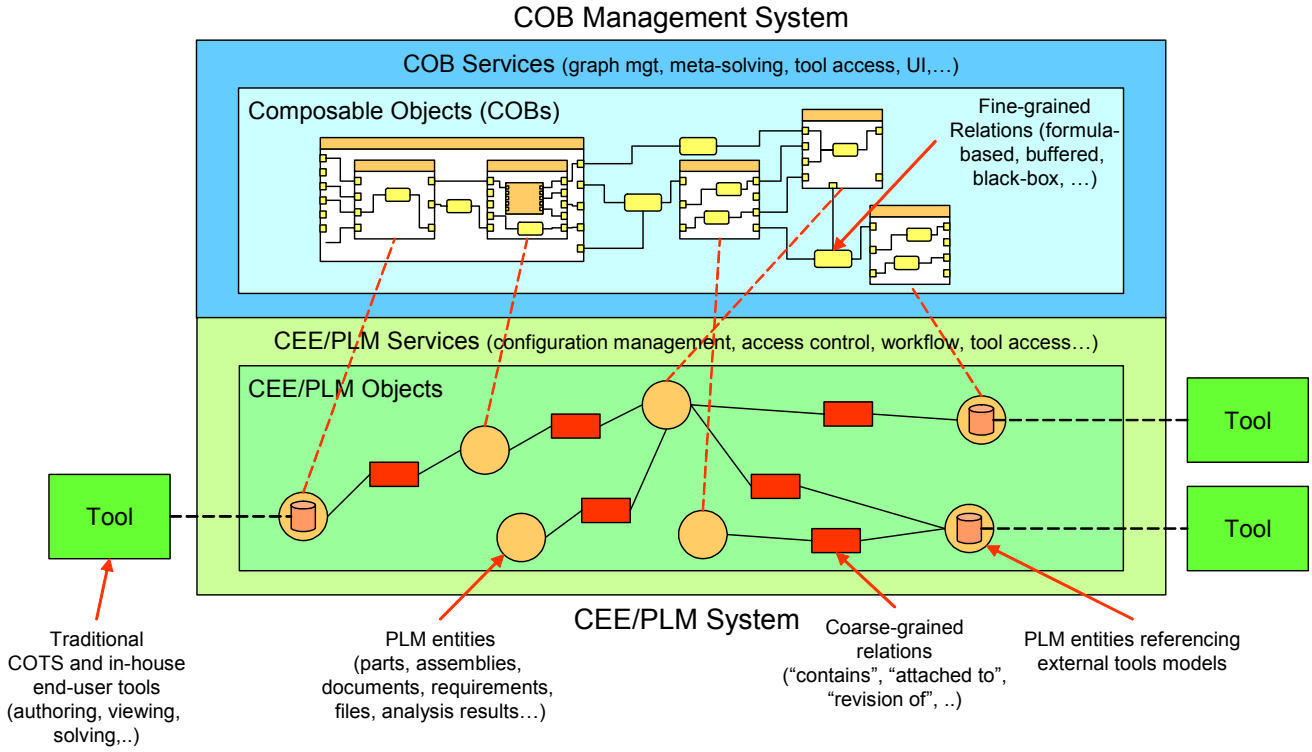


Figure 17: CMS Coupled with a CEE/PLM System
(e.g., the NASA ESMD enterprise-level Windchill-based CEE)

Table 6 below shows a sample of possible solution providers for each COB-based CEE system component in the two figures above (the shaded ones are already developed or currently under development by GIT). This table shows that many components leverage existing COTS capabilities to a large degree, with COBs providing the means to represent fine-grained interconnection knowledge and to orchestrate interoperability among the diverse models associated with these COTS capabilities.

Table 6: Typical Sample CEE Subsystem Solution Providers[5]

Component	Application Category	Providers
CMS Management Client Tools	System Management Utilities	GIT (R&D extensions in progress)
COB-Enabled End-User Applications	SysML Modeling Tools	EmbeddedPlus (EmbeddedPlus Engineering), Rhapsody (I-Logix), RSA (IBM), Studio (Artisan Software), TAU (Telelogic), Teamcenter SE (UGS)
	Requirements Management Tools	Cradle (3SL), DOORS (Telelogic), Eclipse, RequisitePro (IBM), Teamcenter Requirements (UGS)
	PWA Warpage, Chip Packaging, PWB Layup	GIT
CMS	COB Services	GIT (R&D extensions in progress)
Traditional Engineering Applications	ECAD/MCAD	Allegro (Cadence), Expedition (Mentor Graphics), Visual (Zuken); AutoCAD/Inventor (Autodesk), Catia (Dassault), NX (UGS), Pro/E (PTC)
	Discipline-Specific Simulation Tools	Simulink (MathWorks), Modelica, Dymola (Dynasim)
	Mathematical Solvers	Mathematica (Wolfram Research), Matlab (MathWorks)
	CAE: CFD, FEA, ...	Ansys (Ansys Inc.), Flotherm (Flomerics), NX Nastran (UGS), Patran (MSC Software)
	Optimizers, Trade Space Explorers	iSIGHT (Engineous), ModelCenter (Phoenix)
Middleware	Engineering Middleware	AnalysisServer (Phoenix), FIPER (Engineous)
PLM Systems	PLM/PDM	Enovia (Dassault), Teamcenter Engineering/Enterprise (UGS), Vault (Autodesk), Windchill (PTC)

4.1.3 Standards-based Collective Models as a CEE System Component

An important aspect of a CEE system is the conceptual aggregation of all the models that are relevant to a given complex engineering system (e.g., all the models relevant to the space shuttle). We call this aggregation a collective product model or a collective system model [Peak, Lubell, *et al.* 2004]. Figure 18 illustrates how a collective model (outer oval) is generally composed by diverse submodels (inner sets).

[5] Listed alphabetically by tool name.

Each submodel contains information about a specific domain (e.g., Requirements Management, as well as ECAD, MCAD, etc. in later phases). Each tool in this figure typically focuses on viewing or editing a particular type of submodel within this overall system model (e.g., Mentor Graphics CAD tools perform circuit board electrical design and layout, and Pro/E CAD tools develop 3D enclosures and circuit board mechanical assemblies). Portions of the collective model may exist that are not addressed by traditional (COTS) tools (for example, the Cost submodel in the figure). This situation is where so-called "Gap-Filling" tools are required to populate these submodels.

The schemas in this collective model may have been defined by a variety of bodies, including international, government or corporate standard organizations (e.g., STEP APs, IDF, etc.), vendors (UGS' PLM-XML, Dassault Systemes' 3D For All, Autodesk's DWF), may have been custom-defined for the specific application, or may be a combination of standard schemas with custom extensions. In addition, these schemas may be defined using a variety of modeling languages (e.g., OWL, EXPRESS, XML, UML/SysML, etc.). Lastly, the interoperability between each tool and the CEE may be achieved using a variety of technologies (file-based, messaging, SOAP, CORBA, RPC, in-memory, in-process, via a database, etc.).

Realistically, these differences in schemas, modeling languages and interoperability technologies may never be reconciled in a universal standard way. The availability of data exchange standards helps alleviate the problem by providing common schemas that facilitate interoperability among tools and between these tools and the collective model. But any standard is limited in scope and normally targets a specific domain. *Therefore there will always be the need to perform semantic and syntactic mappings between tools and their models to achieve integration and extensibility at a multi-disciplinary level.* Our "multi-technology" approach acknowledges this reality and provides a mechanism for federating models into collective models at a level of abstraction above these differences. We believe engineers can employ SysML as a primary unifying graphical nomenclature to compose federated simulations, while the underlying COBs (via the COB Services) "take care of" (i.e., contain the logic embodied in code) the access to external tools, information retrieval and mapping, and solving for outputs.

As an example, consider a COB that encapsulates requirements information. There will be code behind this COB to access information from a requirements management tool like Cradle and map it to the collective model. Even more desirable, the same COB may access requirements information via a standards-based interface or repository (e.g., via STEP AP233) instead of being dependent on a particular tool like Cradle. By leveraging such standards, the COB automatically also "works" with other requirements management tools that conform to this standard (such as UGS Teamcenter Requirements or Vitech Core).

Overall, this collective model view underscores what we call a model-centric thinking vs. tool-centric thinking. In the latter case, the focus and entry point is a tool which may often seem to hold a model hostage (i.e., forcing you to use that tool to do anything with your model, including not providing you open access to your model). With model-centric thinking, people utilize a variety tools to work on their model (analogous to how machinists utilize a variety machine tools to fabricate their part).

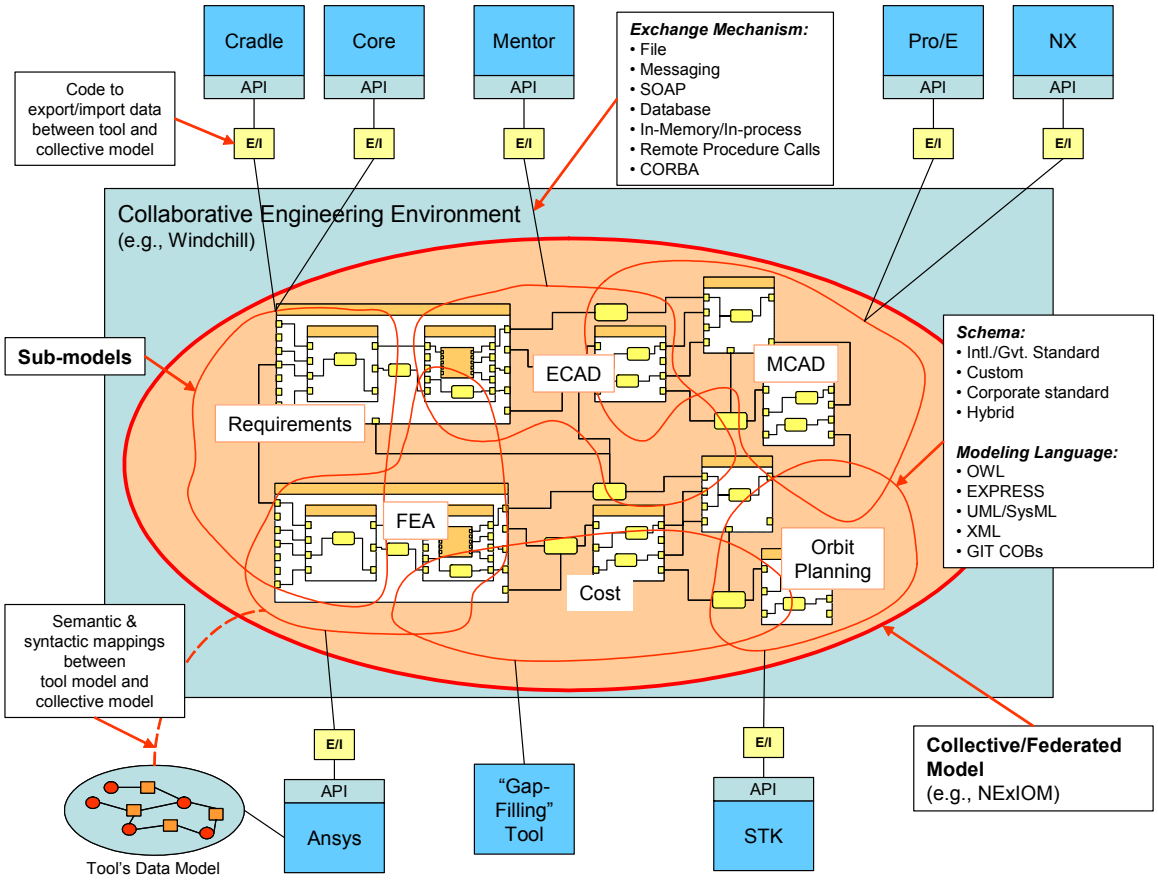


Figure 18: CEE Information Federation via a Standards-based Collective Model

4.2 Composable Objects/Multi-Representation Architecture (COB/MRA) Background

4.2.1 Overview

As mentioned in the previous sections, the COB Representation provides the conceptual foundation of the COB Platform, as it defines the underlying data model and algorithms for the implementation of the COB Platform and its services.

This section overviews the COB Representation — and the related Multi-Representation Architecture (MRA) — and discusses how they relate to the envisioned CEE System. It also provides references for more detailed discussions and example uses of these technologies.

Table 7 below provides a summary of GIT technologies aimed at next-generation CEEs, with brief descriptions of what they are and how they relate to the COB Platform.

Table 7: GIT Technologies for Next-Generation CEEs

Current GIT Technology	What it is	How it relates to the envisioned COB Platform
Composable Objects (COBs) Representation	A constraint graph- and object-based information modeling *representation.* Provides the conceptual foundation for XaiTools.	Will be enhanced to provide the conceptual foundation for the COB Platform.
Multi-Representation Architecture (MRA)	A *design pattern* that represents the primary types of conceptual models in engineering simulation environments and their fine-grain associativities (including idealization relations between design models and analysis models). It has been implemented using COBs and XaiTools for various electronic packaging and aerospace applications to transform and marshal information between external authoring and solving software tools	Will be generalized to systems-of-systems (SoS) and implemented using the COB Platform for the same purpose.
XaiTools	An early *implementation* by GIT of COB and MRA concepts in the form of a *toolkit* of classes, services and an end-user tool (a "COB Browser"). Currently being used in various electronic packaging and aerospace applications developed by GIT.	Will evolve into an implementation of COB Platform and CMS concepts.

4.2.2 Composable Object (COB) Representation

Composable objects (COBs)[6] have been developed by GIT as a means for integrating design models with diverse analysis[7] 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.

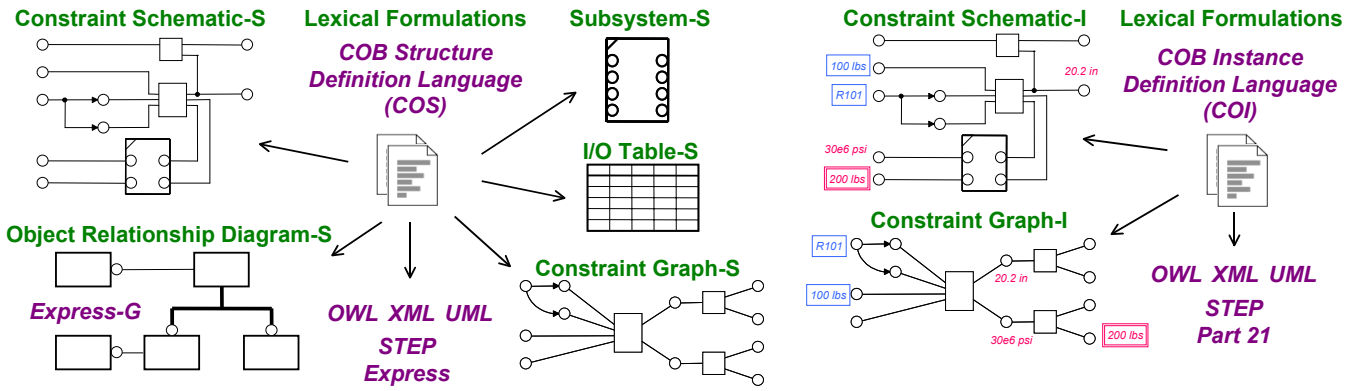
The COB representation is based on object and constraint graph concepts to gain their modularity and multi-directional capabilities. Object techniques provide a semantically rich way to organize and reuse the complex relations and properties that naturally underlie engineering models. Representing relations 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. It has lexical formulations that are computer interpretable, as well as graphical forms that aid human comprehension (Figure 10, Figure 19). For example, the graphical constraint schematic notation (Figure 20) emphasizes object structure and relations among object attributes and has strong electrical schematic analogies. Over the past few years we have been working with other SysML developers to embody COB concepts within SysML (especially regarding its internal block diagram and parametric diagram constructs) [Peak 2002c; Peak 2005]. We believe this approach will a) benefit SysML by providing conceptual formalisms and a broad variety of examples, and b) benefit COBs by leveraging a richer set of UML2-based constructs and broader commercial support by multiple vendors.

[6] COBs are referred in some of the older literature as *Constrained* Objects. The change in name was to better reflect the composability nature of these objects.

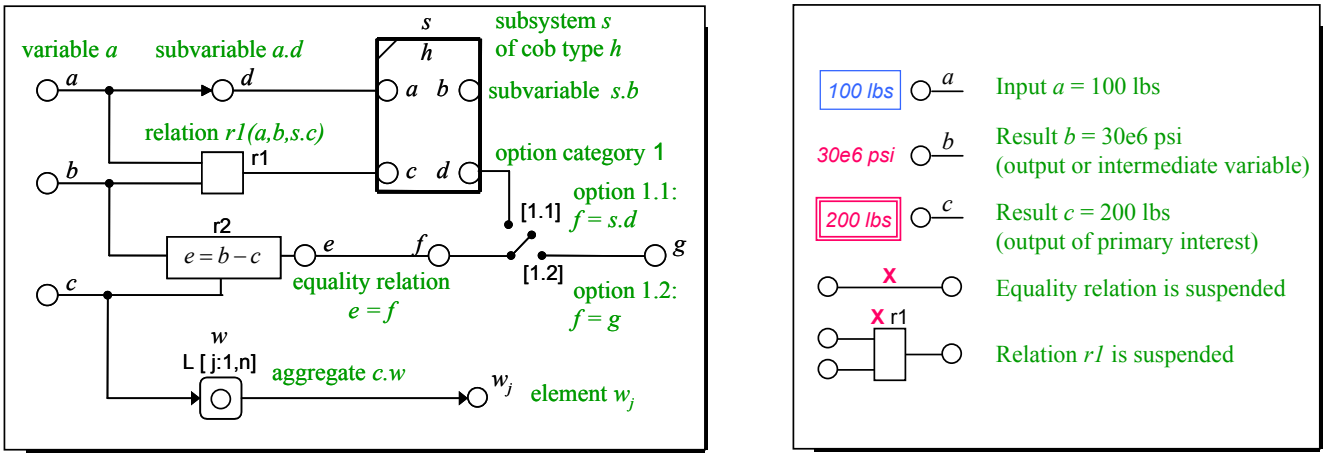
[7] In this overview, "analysis" and "simulation" denotes modeling physical behavior such as stress and temperature. Envisioned next-generation extensions include generalizations for broader classes of modeling and simulation.

Figure 21 provides examples of the main classical COB formulations for a triangle template and its usage in a prism. A prism instance is also shown. See [Peak 1999a, 1999d, 2002c] and [Wilson 2000, 2001] for more details on the COB Representation and further examples.



a. COB structure languages. b. COB instance languages.

Figure 19: Lexical and Graphical Formulations of the COB Representation



a. Structure notation (-S) b. Instance notation (-I)

Figure 20: Basic Constraint Schematic Notation (green text = explanation)

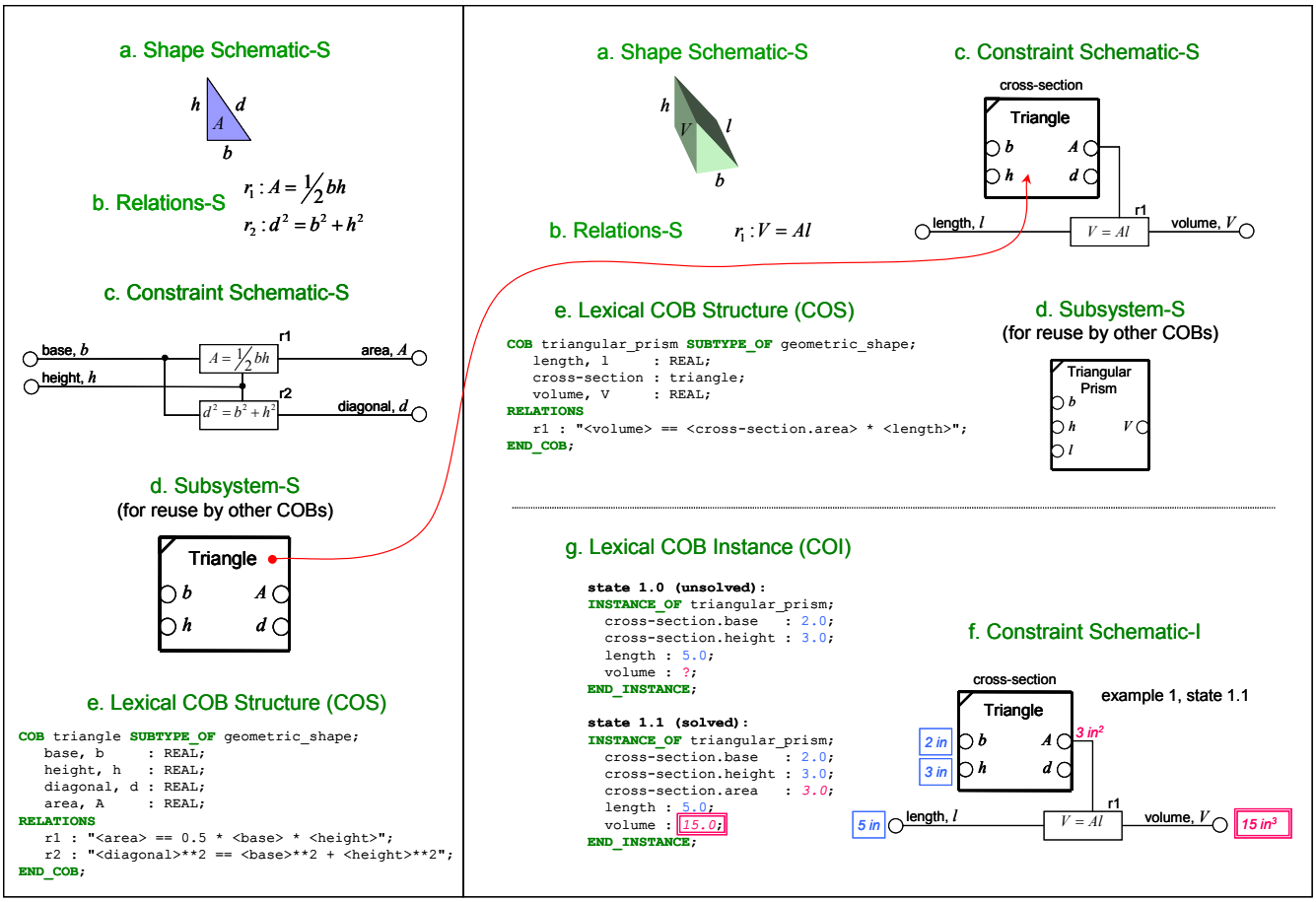


Figure 21: COB Representation Tutorial for Triangles and Prisms

4.2.3 The Multi-Representation Architecture (MRA)

The Multi-Representation Architecture (MRA) (Figure 22) is the conceptual foundation of an X-analysis integration (XAI)[8] methodology based on ontological patterns that naturally exist in engineering analysis processes. It is particularly aimed at design-analysis integration in CAD/CAE environments with high diversity (e.g., diversity of parts, analysis discipline, analysis idealization fidelity, design tools, and analysis tools) and where explicit design-analysis associativity is important (e.g., for automation, knowledge capture, and auditing). In this context, analysis means simulating the physical behavior of a part or system (e.g., determining the stress in a circuit board solder joint).

[8] X = design, manufacture, sustainment, and other lifecycle phases.

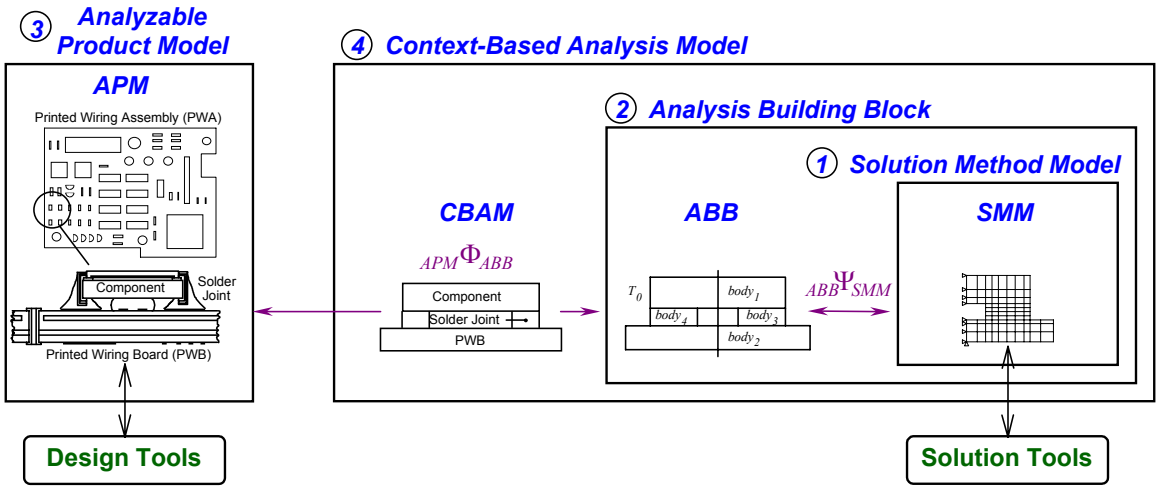


Figure 22: The Multi-Representation Architecture (MRA)

The MRA contains intermediate representations as stepping stones to achieve the flexibility and modularity dictated by complex domains like simulation-based design and engineering (SBD/SBE). Employing an extended object-oriented approach, these intermediate representations are naturally groupings of concepts that occur between traditional design and analysis models. The MRA is particularly aimed at capturing reusable analysis knowledge at the preliminary and detailed design stages.

The MRA conceptual patterns (Figure 22) include the following (all of which are represented as COBs):

- *Analyzable product models (APMs):* Represent knowledge-based design models augmented with analysis-oriented overlays. Include multi-fidelity idealizations and multi-source design information coordination (including interfacing with diverse CAD tools and design-oriented descriptive resources).
- *Context-based analysis models (CBAMs*): Represent product-specific analysis modules/templates. Capture idealization decisions inside CAD-CAE associativity relations. Connect APMs and ABBs.
- *Analysis building blocks (ABBs):* Represent product-independent analytical concepts as semantically rich reusable, modular, tool-independent objects. Generate SMMs based on solution technique-specific considerations such as symmetry and mesh density.
- *Solution method models (SMMs):* Represent solution method-specific models. Support white box reuse of existing tools (e.g., FEA tools and in-house codes). Automatic interactions occur through native command lines and/or APIs based on standards like CORBA and SOAP.

The reader is referred to [Peak 1998, 1999a, 1999d, 2002a] and [Tamburini 1997a, 1999] for more details on the MRA and examples.

4.2.4 Towards a Next-Generation MRA for Systems-of-Systems (SoS)

Table 8 below summarizes the major types of patterns that exist when complex systems are described and simulated. The first column shows terminology developed in our original MRA work where the focus was on patterns for domain-level design-analysis integration (DAI). The second column highlights the purpose of each traditional pattern. The last column provides terminology towards generalizing these concepts for the modeling and simulation of arbitrary systems-of-systems (SoS). We plan to further develop this generalized MRA approach for such applications in future phases of this work.

Table 8: Generalized MRA Patterns for Complex Systems Modeling & Simulation (M&S)

Traditional Patterns (for CAD-CAE)	Traditional CAD-CAE Purpose regarding Design-Analysis Integration (DAI)	Generalized Patterns (for complex systems)
design tools (CAD)	- Define systems (parts, assemblies, …) in necessary & sufficient descriptive terms (not behavioral) - Usually are COTS tools	system description tools
analyzable product models (APMs)	- Represent design aspects of products and enable connections with design tools - Support idealizations usable in numerous analysis models - Have possibly many associated CBAMs that verify requirements	integrated system model
context-based analysis models (CBAMs)	- Contain linkages explicitly representing design-analysis associativity, indicating usage of APM idealizations - Create analysis models from ABBs and automatically connect them to APM attributes - Represent common analysis models as automated, predefined templates - Support interaction of analysis models of varying complexity and solution method - Enable parametric design studies via multi-directional input/output (in some cases)	context-based simulation model *(system-specific simulation model)*
analysis building blocks (ABBs) *(generic analytical concepts)*	- Represent analytical concepts as composable objects - Act as semantically rich 'pre-preprocessor' & 'post-postprocessor' models. - ABB instances create SMM instances based on solution method considerations and receive results after automated solution tool execution	simulation building block *(generic analytical concepts)*
solution method models (SMMs)	- Packages solution tool inputs, outputs, and control as integrated objects (often as a componentized wrapping of solution tool native files) - Automates solution tool access and results retrieval via tool agents and wrappers	simulation method model
solution tools (CAE)	- Execute simulation models (often as vendor-specific native files) - Usually are COTS tools	simulation tool (solver)

version: 2005-10-26

4.2.5 XaiTools - a Reference Implementation

To help validate the COB Representation and MRA technique, we have developed COBs for a variety of aerospace and electronic packaging test cases using our toolkit called *XaiTools*. *XaiTools* is an example embodiment of COB concepts and includes an API, samples and a spreadsheet-like COB Browser (Figure 23) which supports knowledge capture in a non-causal object-oriented manner. We use constraint management techniques to employ existing solvers whenever possible such as commercial math and finite element analysis (FEA) tools (e.g., Ansys). Tutorials and short courses convey the concepts and include examples that combine generic and domain-specific COBs. COBs have been deployed in production usage environments to help automate chip package thermal resistance analysis. See [Wilson 1999, 2000] for more details on *XaiTools*.

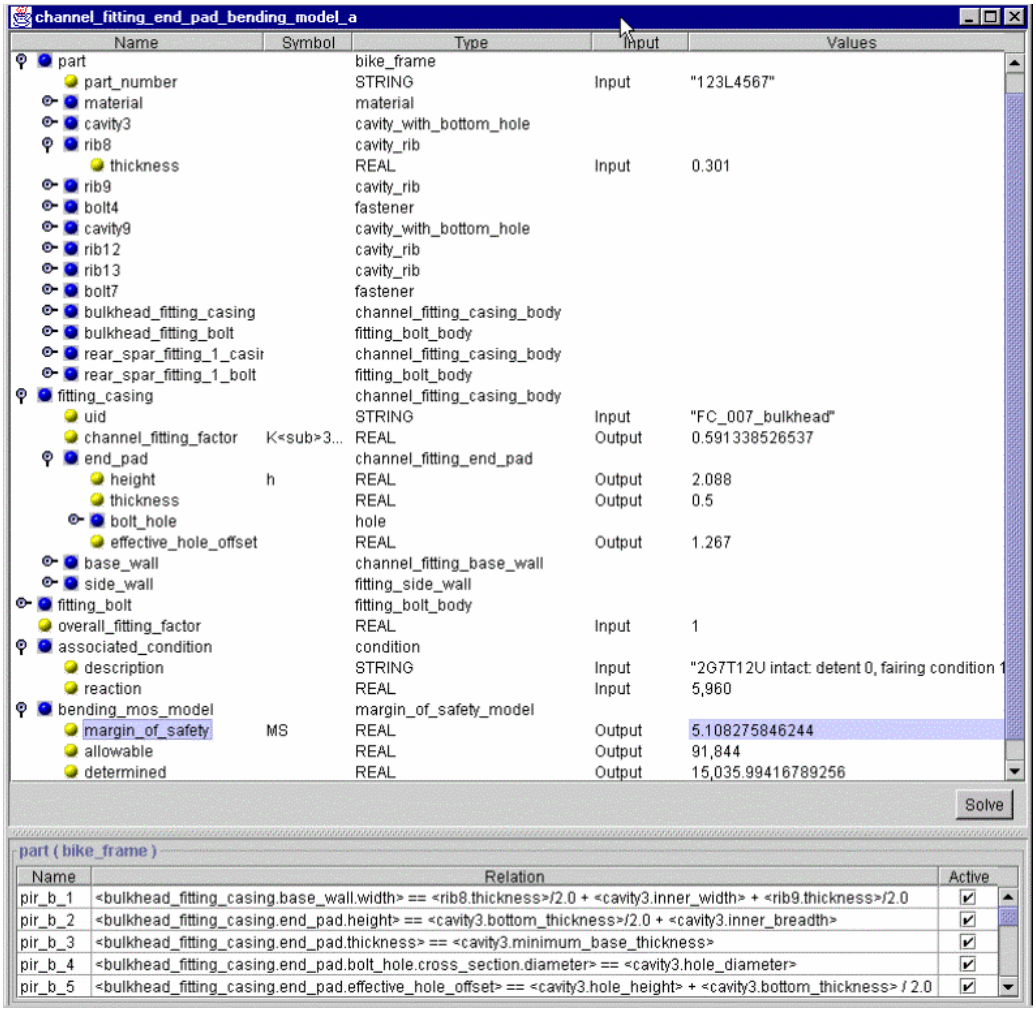


Figure 23: XaiTools COB Browser

4.2.6 Space System Example

See Appendix A for a conceptual test case illustrating this COB/MRA approach (including usage of SysML) for a satellite system example known as FireSat.

4.3 COB Representation Requirements to Enable Next-Generation CEEs

So far we have discussed the challenges of Collaborative Engineering (Section 3.1) and the desired capabilities of a CEE System to address these challenges (Section 3.2). In Section 4.1 we discussed how these capabilities are covered by each subsystem of our COB Platform. We also established that the COB Representation is the conceptual foundation for the COB Platform.

Now that we have a working knowledge of the COB Representation, in this section we list the requirements that the COB Representation must satisfy to support the promised COB Platform functionality, and thus enable next-generation CEE systems.. These requirements are presented in Table 9 below.

Table 9: COB Representation Required Capabilities

Capability	Req #	COB Representation Requirement
CAP-001 (End-user tools for collaborative Systems Engineering)	CR-001	The COB Representation's data model and operations shall be exposed via an API to enable development of COB-based applications.
	CR-002	The COB Representation shall provide the basic constructs (or "building blocks") for defining the components of a system; including systems, subsystems, system parameters, parameter constraints, relations between and within subsystems, and relations between system parameters .
	CR-003	The COB Representation shall provide mutually equivalent lexical and graphical representations for these constructs.
	CR-004	The COB Representation shall support interactive constraint schematic construction, hybrid graphs (mixing causal and non-causal relations), and automated effective inversion of causal relations.
CAP-002 (Common graphical notation for modeling and simulating systems)	CR-005	The COB Representation shall specify how its constructs shall be represented in SysML, and provide workarounds whenever there is no direct mapping between the two.
	CR-006	The COB Representation shall be able to interoperate with the data exchange format chosen by the SysML standard (still TBD, but most likely to be XMI)
CAP-003 (Constructs for modeling complex system relations)	CR-007	The COB Representation shall enable the representation of the parametric relationships among all fidelity levels (coarse, detailed), domains (mechanical, electrical, software, controls, optics, thermal, etc.) and types (physics-based, functional, analytical, simulation, visualization, design, etc.) of system and component models.
	CR-008	The COB Representation shall provide constructs for defining *formula-based* relations (see CAP-003 in Section 3.2).
	CR-009	The COB Representation shall provide constructs for defining *equality* relations (see CAP-003 in Section 3.2).
	CR-010	The COB Representation shall provide constructs for defining *constraint* relations (see CAP-003 in Section 3.2).
	CR-011	The COB Representation shall provide constructs for defining *aggregate* relations (see CAP-003 in Section 3.2).
	CR-012	The COB Representation shall provide constructs for defining *buffered* relations (see CAP-003 in Section 3.2).
	CR-013	The COB Representation shall provide constructs for defining *selector* relations (see CAP-003 in Section 3.2).
	CR-014	The COB Representation shall provide constructs for defining *breaker* relations (see CAP-003 in Section 3.2).
	CR-015	The COB Representation shall provide constructs for defining *black-box* relations (see CAP-003 in Section 3.2).
	CR-016	The COB Representation shall provide constructs for defining *unidirectional* relations (see CAP-003 in Section 3.2).
	CR-017	The COB Representation shall allow the definition of the possible *directions* in a relation (i.e., allowable sets of inputs and outputs).

	CR-018	The COB Representation shall provide constructs for capturing the uncertainty of a relation.
	CR-019	The COB Representation shall support aggregate attributes (i.e., attributes of type List)
CAP-004 (End-user tools to visualize the complex interconnections in a system)	CR-020	The COB Representation shall enable a user to identify, represent, visualize, and navigate the relationships among the diverse types of models used to analyze and simulate systems.
	CR-021	The COB Representation shall provide equivalent computable lexical forms and human-interpretable graphical forms for representing complex constraints and relations between systems.
	CR-022	The COB Representation shall provide an intuitive visualization language for rigorously specifying non-causal and algorithmic relationships between physical assembly parameters and corresponding idealized analysis model parameters.
	CR-023	The COB Representation shall provide a mechanism to determine what system components (systems, parameters or relations) are affected by a change in a given component.
	CR-024	The COB Representation shall provide a mechanism to measure the *slack* of the relations in the system (see CAP-004 in Section 3.2).
	CR-025	The COB Representation shall provide a mechanism to determine what parameters are bound (constrained) as a consequence of binding a given parameter. Conversely, it shall provide a mechanism to determine what parameters should be bound in order to bind another parameter.
CAP-005 (An approach to manage consistency of shared parameters throughout the project lifecycle)	CR-026	The COB Representation shall provide a mechanism to continuously check that the system is in a consistent state and the constraints and relations are not being violated as users make changes and additions to the system.
CAP-006 (Versioning and configuration control mechanism)	CR-027	The COB Representation shall provide constructs for capturing versioning of system components (systems, subsystems and relations).
	CR-028	The COB Representation shall provide constructs for capturing versioning of simulations sets (which include the set of inputs, the versions of the system and design models, and all the run-time selections made).
	CR-029	The COB Representation shall provide the necessary check in/check out logic to ensure that the consistency of systems is maintained throughout versioning.
CAP-007 (Ability to create libraries of reusable components)	CR-030	The COB Representation shall allow the definition of reusable, adaptable analysis building blocks.
	CR-031	The COB Representation shall support inheritance of attributes and relations.
CAP-008 (Ability to capture the assumptions, rationale and limitations of a model)	CR-032	The COB Representation shall provide constructs for capturing the assumptions, rationale and limitations of a model.
	CR-033	The COB Representation shall provide a mechanism for filtering the models that are applicable to a particular context, based on its assumptions, rationale and limitations.
	CR-034	The COB Representation shall provide a mechanism to capture the design intent while using models of multiple fidelities (for example, capturing that a 2D model and 3D model of a component are being used for the same intent)

CAP-009 (Simulation orchestration)	CR-035	The COB Representation shall be independent of the specific CAD/E tools used to create, manage, and compute these models, since tools for different domains are often provided by different vendors.
	CR-036	The COB Representation shall support the use of external programs as white-box relations.
	CR-037	The COB Representation shall provide a framework to interoperate with COTS and interoperability middleware.
	CR-038	The COB Representation shall be able to *leverage* standard representations of product data (such as ISO STEP) but shall not *rely* on their availability or completeness. In other words, the methodology shall also work with custom/ad-hoc/non-standard representations or *extensions* to standard representations.
	CR-039	The COB Representation shall support providing *relations* (parametric constraints) as outputs of a relation in addition of numeric values.
	CR-040	The COB Representation shall support the use of value range constraints to remove extraneous solutions when there are multiple solutions for a given parameter.
	CR-041	Whenever possible, the COB Representation shall leverage and/or be able to interoperate with other methodologies (such as XML/XMI, UML/SysML, OWL, Semantic Webs, STEP, topic maps).
CAP-010 (Requirements allocation and traceability)	CR-042	The COB Representation shall provide constructs for defining requirements (quantitative or otherwise) and allocating them to the components of the system implementing them.
	CR-043	The COB Representation shall provide a mechanism for checking conformance to quantitative requirements.
	CR-044	The COB Representation shall provide the mechanism for figuring out which requirements can be relaxed in the event of a conformance conflict.
CAP-011 (Workflow)	CR-045	The COB Representation shall provide constructs for *defining* workflows.
	CR-046	The COB Representation shall provide a mechanism for *executing* workflows (i.e., advancing a model of a system through a sequence of steps, each of which may query and/or change the values of the system parameters).

5 Summary

We presented a vision for next-generation collaborative engineering environments (CEEs) that are based on the composable object (COB) knowledge representation. This methodology leverages the multi-representation architecture (MRA) for simulation templates, the user-oriented SysML standard for system modeling, and standards like STEP AP233 (ISO 10303-233) for enhanced interoperability.

The objective of this document has been to define requirements for the COB representation. However, to achieve that objective, we first documented today's major challenges and pain points of CEEs and then mapped these challenges to desired CEE system capabilities. Then we described an advanced CEE methodology in terms of envisioned COB components. Given that basis, we could then effectively specify COB requirements.

In our current project we are defining and developing next-generation COB capabilities. Progress to date includes the following accomplishments:

- We implemented various COB examples as SysML models in a representative commercial modeling tool (Artisan Studio). See draft space system examples in Appendix A, plus other more complete examples for mechanical, electrical, and hydraulic systems [Peak *et al.* 2005; Peak, 2005].
- We have implemented a prototype interface between the above SysML tool (Studio) and *XaiTools*, which enables SysML-based model authoring and COB-based execution using commercial math and FEA solvers. We are also in the process of implementing a similar interface to a dynamics system modeling tool (Dymola).
- To increase ease-of-use and familiarity, we have extended COB lexical support in *XaiTools* to include XML-based formats.

From these experiences it appears that SysML will be able to provide most, if not all, of the structural representation constructs imposed by the above COB requirements. Additionally, we have identified subsolving constraint graph algorithms that will likely form the basis for the associated COB algorithm extensions.

Given these promising results thus far, we are optimistic we will fulfill the COB requirements defined in this document in subsequent phases and thus provide the foundation for next-generation CEEs.

References

Aughenbaugh, J. M., and Paredis, C. J. J. (2004) "The Role and Limitations of Modeling and Simulation in Systems Design," in *Proceedings of the 2004 ASME International Mechanical Engineering Congress and RD&D Expo*, paper no. IMECE2004-5981, Anaheim, CA, November 13-19,. http://www.srl.gatech.edu/Members/jaughenbaugh/papers_presentations/IMECE2004-5981.pdf

EIS Lab (2005) The Composable Object (COB) Knowledge Representation: Enabling Advanced Collaborative Engineering Environments (CEEs), Project Web Page, Georgia Tech Engineering Information Systems Lab, http://eislab.gatech.edu/projects/nasa-ngcobs/

Kevin L.G. Parkin , Joel C. Sercel , Michael J. Liu , Daniel P. Thunnissen (2003) ICEMaker™: An Excel-Based Environment for Collaborative Design" http://monolith.caltech.edu/Papers/Parkin%20IEEE%20Paper%201564.pdf

Koch, P. (2002). "Probabilistic Design: Optimizing for Six Sigma Quality" *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, Colorado, Apr. 22-25. AIAA-2002-1471.

Larson WJ and Wertz JR ed. (1999) Space Mission Analysis and Design (SMAD), 3rd Ed. Microcosm, Inc.

NASA (1995). "NASA Systems Engineering Handbook (SP-610S)."

Paredis, C.J.J., A. Diaz-Calderon, R. Sinha, and P.K. Khosla (2001) "Composable Models for Simulation-Based Design", *Engineering with Computers*. Vol. 17, pp. 112-128.

Peak R.S., Fulton R.E., Nishigaki I., Okamoto N. (1998) Integrating Engineering Design and Analysis Using a Multi-Representation Approach. *Engineering w. Computers*, 14 (2) 93-114.

Peak R.S. *et al,* (1999a) Towards the Ubiquitization of Engineering Analysis to Support Product Design. Invited Paper for Special Issue: Advanced Product Data Management Supporting Product Life-Cycle Activities, *Intl. J. Comp. Applications Technology* 12(1): 1-15.

Peak, R. S., Scholand, A. J., Tamburini, D. R., Fulton, R. E. (1999d) "Towards the Routinization of Engineering Analysis to Support Product Design," Invited Paper for Special Issue: Advanced Product Data Management Supporting Product Life-Cycle Activities, *Intl. J. Computer Applications in Technology*, Vol. 12, No. 1, 1-15, 1999.

Peak R.S. (2002a) Techniques and Tools for Product-Specific Analysis Templates Towards Enhanced CAD-CAE Interoperability for Simulation-Based Design and Related Topics. Intl. Conf. Electronic Packaging. JIEP/ IMAPS Japan, IEEE CPMT. Tokyo.

Peak R.S. (2002c) 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, R. S. (2003) Characterizing Fine-Grained Associativity Gaps: A Preliminary Study of CAD-E Model Interoperability. ASME DETC Paper No. CIE-48232, Chicago. http://eislab.gatech.edu/pubs/conferences/2003-asme-detc-peak/

Peak R.S., J Lubell, V Srinivasan, SC Waterbury (Dec. 2004) STEP, XML, and UML: Complementary Technologies. Product Lifecycle Management (PLM) Special Issue. *J. Computing & Information Science in Engineering*.

Peak R.S., S. Friedenthal, A. Moore, R. Burkhart, S. Waterbury, M. Bajaj, I. Kim (2005) Experiences Using SysML Parametrics to Represent Constrained Object-based Analysis Templates. 2005 NASA-ESA Workshop on Product Data Exchange (PDE), Atlanta. http://eislab.gatech.edu/pubs/conferences/2005-pde-peak/

Peak R.S. (2005) "SysML Emphasis at GIT" and "GIT SysML Parametrics Work". Update to OMG SE DSIG, Atlanta. http://eislab.gatech.edu/pubs/seminars-etc/2005-09-omg-se-dsig-peak/

SysML Forum (http://www.sysml.org/)

Tamburini, D. R., Peak, R. S., Fulton, R. E. (1997a) "Driving PWA Thermomechanical Analysis from STEP AP210 Product Models," ASME Intl. Mech. Engr. Congress & Expo, *CAE/CAD and Thermal Management Issues in Electronic Systems*, EEP-Vol. 23/HTD-Vol. 356, Agonafer, D. et al., Dallas, 33-45.

Tamburini, D.R (1999), The Analyzable Product Model Representation to Support Design-Analysis Integration. Ph.D. Thesis - Mechanical Engineering. 1999, Atlanta: Georgia Institute of Technology. http://eislab.gatech.edu/pubs/theses/99tamburini/

Wilson, M. W., Peak, R. S., Tamburini, D. R. (1999) *XaiTools Users Guide*. EIS Lab, Georgia Institute of Technology, Atlanta.

Wilson, M. W.(2000), The Constrained Object (COB) Representation for Engineering Analysis Integration, Masters Thesis, Georgia Institute of Technology, Atlanta.

Wilson M., Peak R., Fulton R. (June, 2001) Enhancing Engineering Design and Analysis Interoperability - Part 1: Constrained Objects. First MIT Conference Computational Fluid and Structural Mechanics (CFSM), Boston. *Received Young Researcher Fellowship award.*

Appendix A - Satellite System Example: FireSat

This Appendix highlights how COBs and their embodiment as SysML models might be applied to a sample space system. We utilize the FireSat fire-detecting satellite system described in *Space Mission Analysis and Design* (SMAD) [Larson and Wertz, 1999] as the backdrop.

First, Figure 24 overviews the SMAD space system design process. Our examples below start roughly at the "Definition of Elements" level and then proceed to the conceptual design of a sample subsystem: the attitude determination and control (ADC) system. We end with several leaf-level domain models: design and simulation models for an ADC subsystem circuit board (which are themselves composed from generic reusable building blocks).

Figure 25 is an abstract schematic of the FireSat system showing estimates for key parameters like lifetime, orbit, and mass.

Figure 26 is a conceptual draft of this test case. It shows the FireSat system design (from SMAD Chapter 11) represented as SysML parametric diagrams for three levels of COB-based models. Figure 26 (a) shows the top-level system design where items like orbit and mass properties are modular templates utilized by several of the subsystems. Figure 26 (b) is an initial design of the ADC subsystem where the relations and parameters needed to support magnetic torquer subsystem design have been included. Figure 26 (c) goes one level deeper to illustrate initial magnetic torquer subsystem sizing and design.

This magnetic torquer system has a current drive electronics subsystem, which eventually decomposes to assemblies including circuit boards. Figure 27 (a) is a COB-based MRA panorama depicting design and simulation models at multiple abstraction levels that are all utilized at this same leaf-level of system decomposition. Figure 27 (b) is a SysML class diagram view for the printed wiring board aspects of this panorama (PWB = bare circuit board).

Figure 27 (c) and Figure 27 (d) are SysML parametric templates for two of these physics-based simulations: pwb_extensional_model and pwb_1D_warpage_model. Both of these are product-specific templates known as context-based analysis models (CBAMs) in terms of the MRA. Figure 27 (a) and Figure 27 (c) show how the design aspects come from a domain tool (ECAD) and other design sources coordinated by an analyzable product model (APM). The template connects these design aspects to a generic analytical building block (the deformation model block), which is processed as a solution method model (SMM) using general purpose solvers like FEA tools. Decomposed requirements are the prime motivation behind such templates, and they typically provide both condition/environment inputs like temperature and results evaluation factors like margin of safety as shown in the diagram.

The template in Figure 27 (d) has a similar structure. Its deformation model is composed of the extensional rod analysis building block (ABB) seen in Figure 28, which bottoms out in another ABB (a linear elastic material model) that contains only primitive attributes and relations. Such ABBs are often solved using external COTS math solvers. Classical COB constraint schematic views (a motivator behind this new SysML diagram notation) of these same ABBs were given in Figure 10.

Altogether, this example covers diverse interconnected models spanning roughly 6 levels of system decomposition (from top-level satellite system to circuit board features) and 7 levels of abstraction (including from native CAD model to context-specific simulation model to generic simulation building blocks to native solver tool). We hope to implement such an example in the near future as a test case to verify and validate the COB/MRA-based approach described in this document.

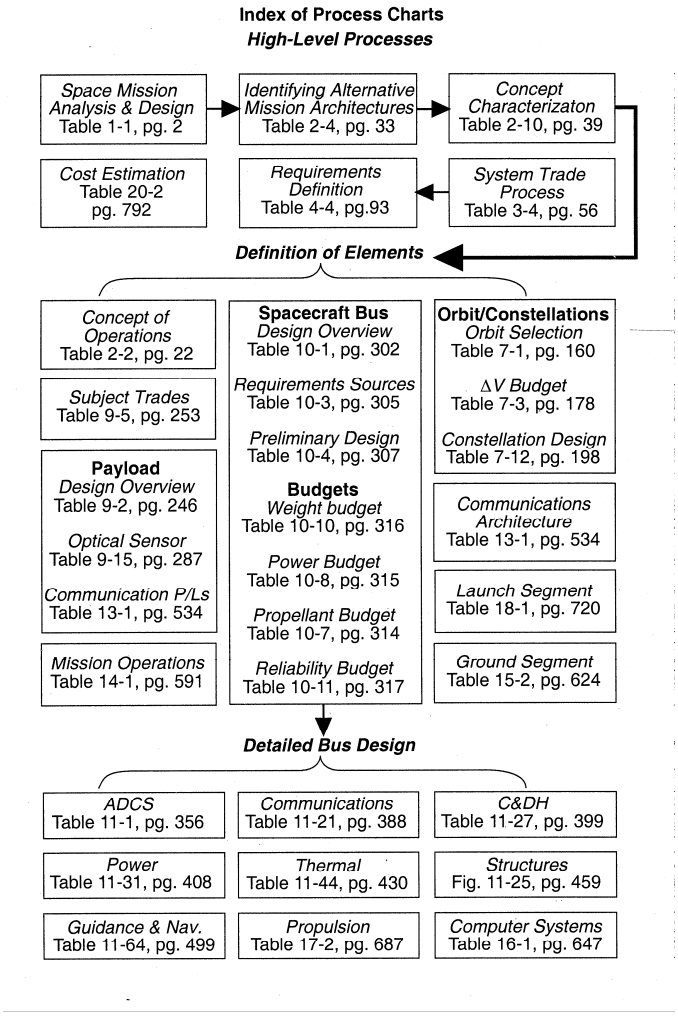


Figure 24: Space System Design Process [Larson and Wertz, 1999]

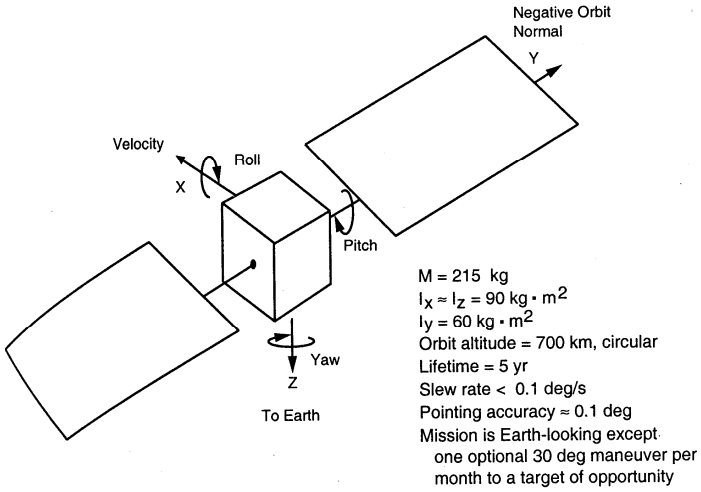


Figure 25: FireSat Space Satellite Abstract System Schematic [Larson and Wertz, 1999]

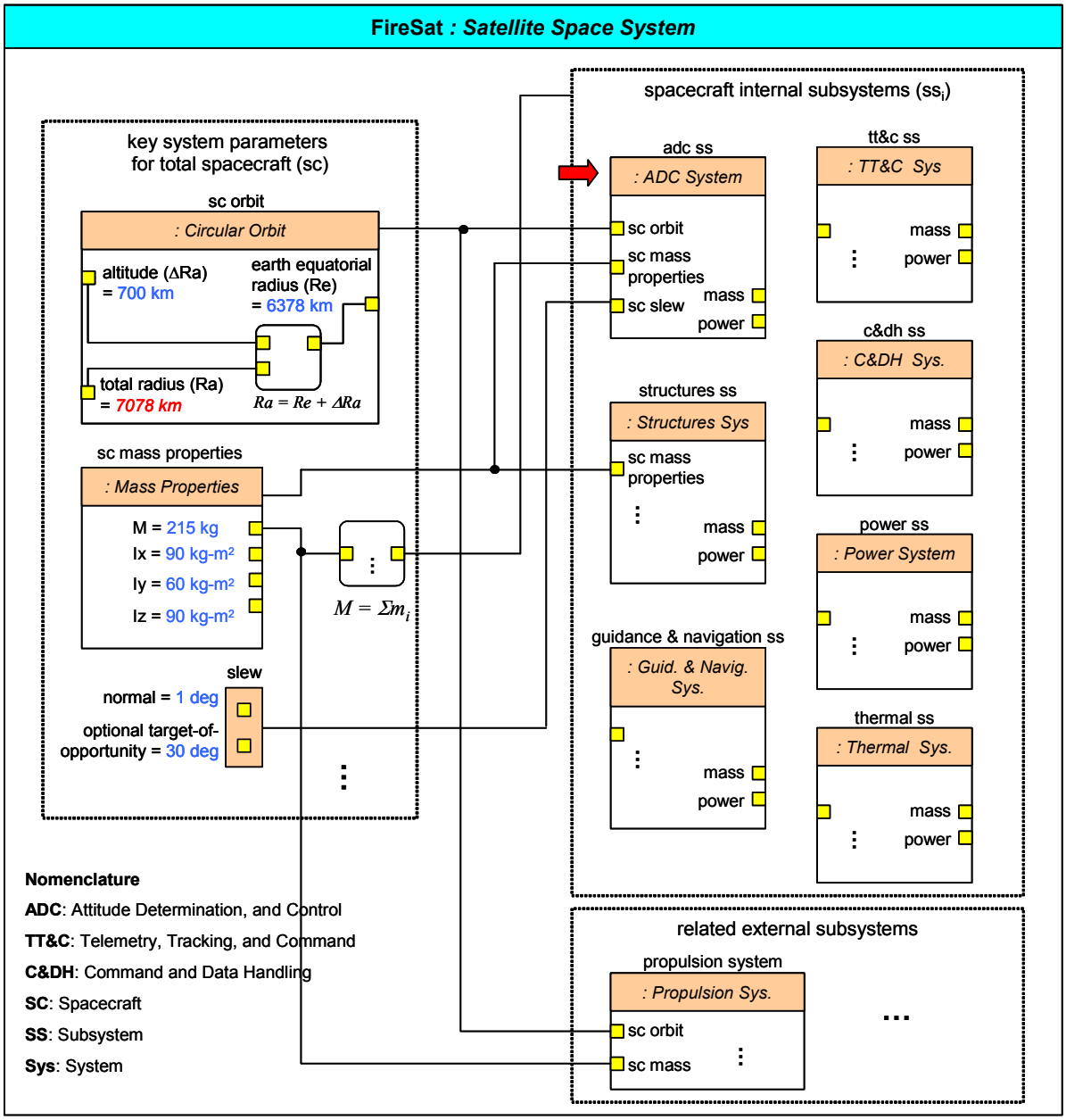


Figure 26 (a): FireSat top-level system design

Figure 26: FireSat System Design as COB-based SysML Diagrams (Conceptual Draft)

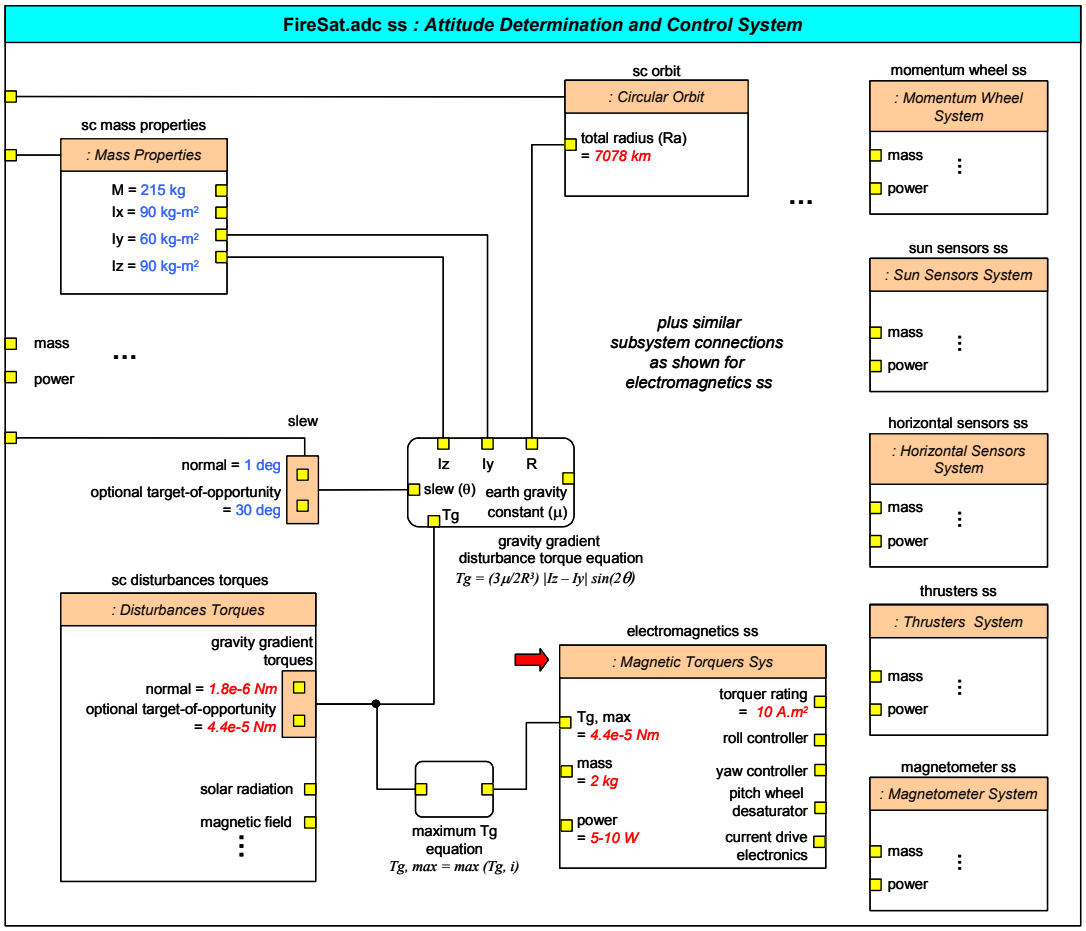


Figure 26 (b): FireSat ADC subsystem design (initial subsystem requirements and sizing)

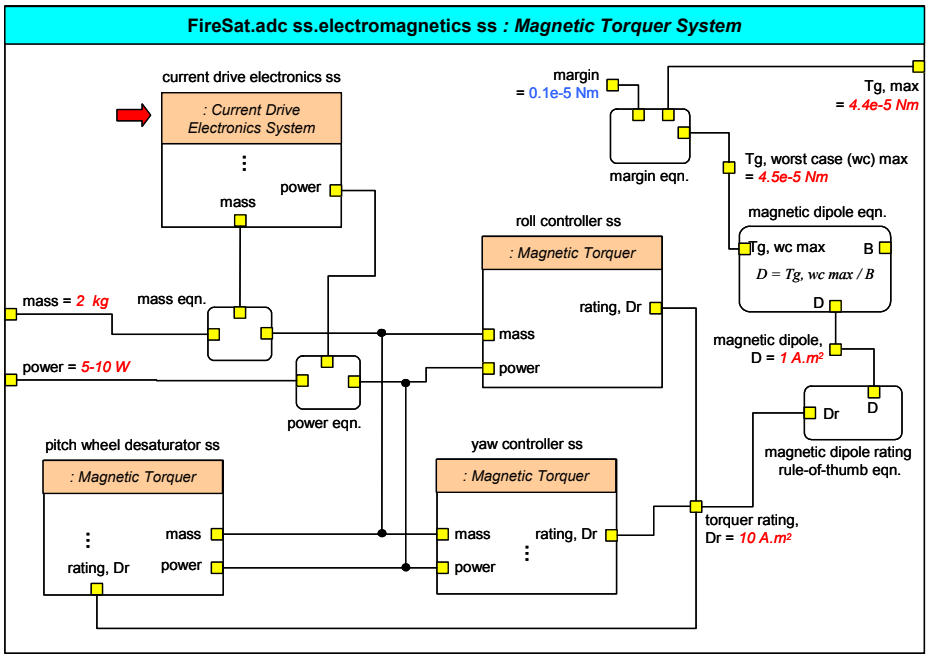


Figure 26 (c): FireSat magnetic torquer subsystem design (initial subsystem requirements and sizing)

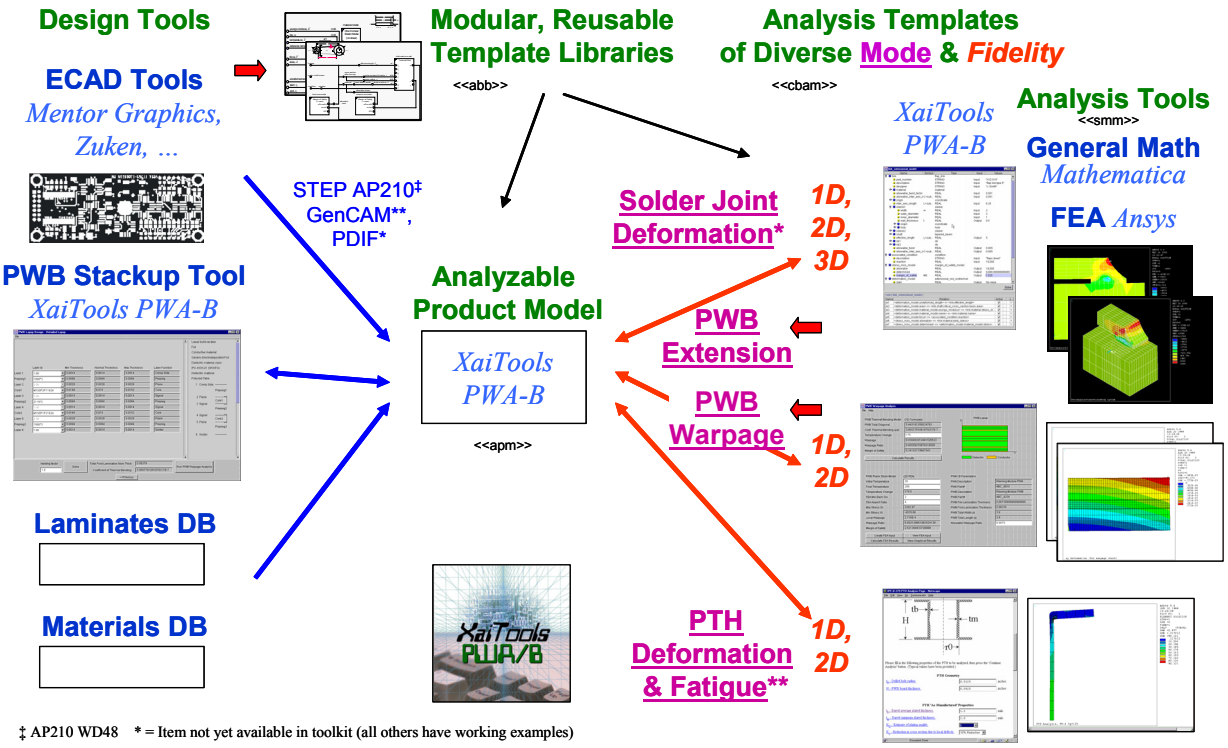


Figure 27(a): Design-Analysis Interoperability Panorama

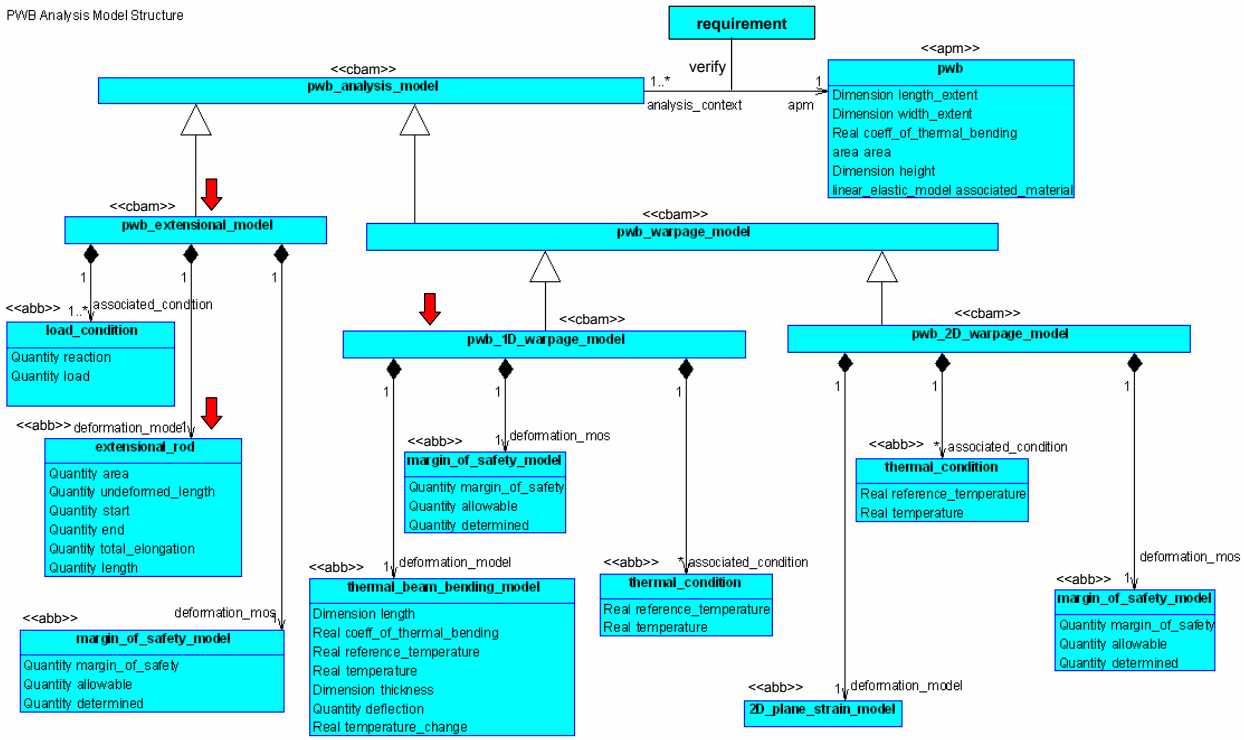


Figure 27 (b): Circuit Board Analysis Template Structure (UML/SysML Class Diagram)

Figure 27: COB/MRA-based Design and Simulation Templates for a Leaf-level FireSat Subsystem: Circuit Boards

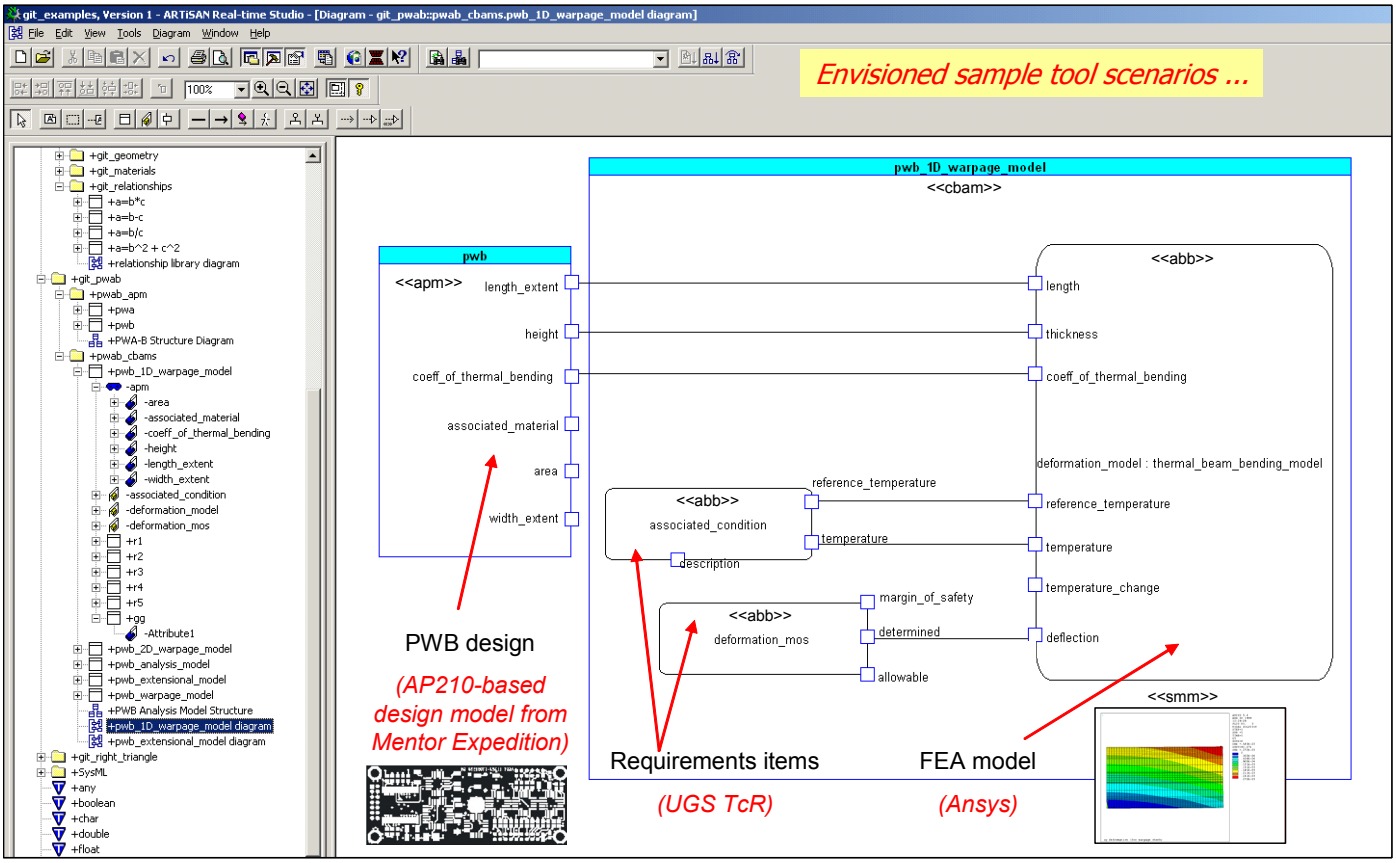


Figure 27 (c): Sample SysML-Based Circuit Board Analysis Template: pwb_1D_warpage_model (Implemented in the Artisan Studio UML/Modeling Tool)

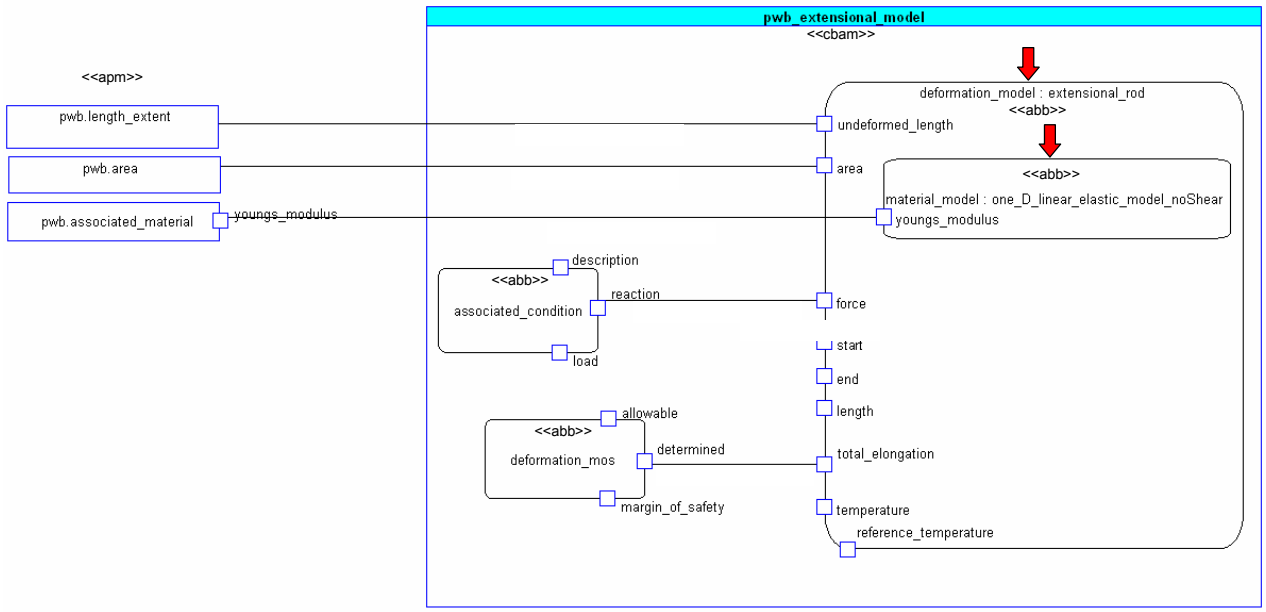


Figure 27 (d): Sample SysML-Based Circuit Board Analysis Template: pwb_extensional_model

Figure 27 (continued)

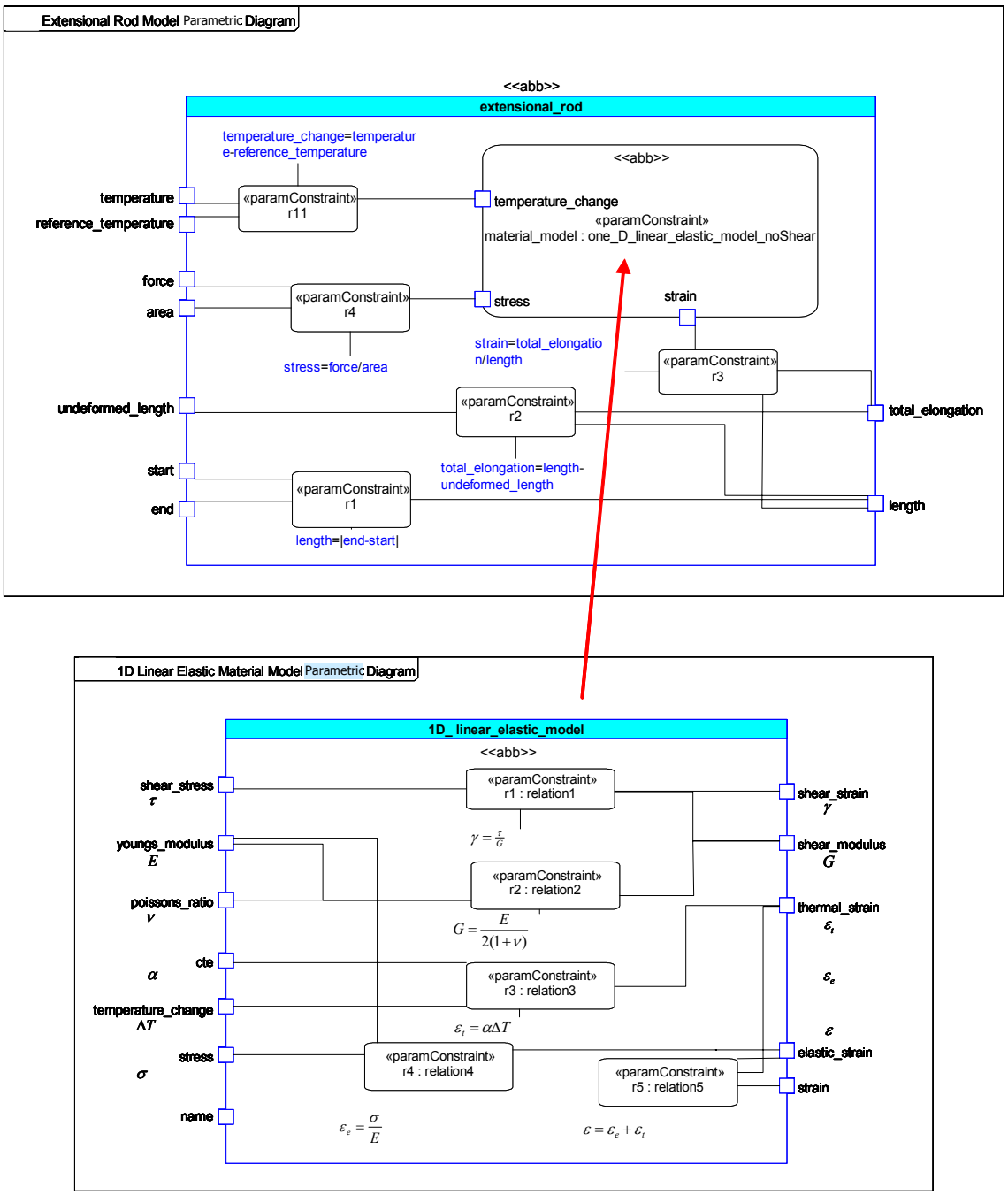


Figure 28: General Purpose Analysis Building Blocks (ABBs) Utilized in Figure 27 (d) (SysML Formulations of ABBs in Figure 10)