A System Engineer's Position on the Unified Modeling Language

J. F. Skipper, PhD.
Senior Systems Engineer
Vitech Corporation
www.vitechcorp.com

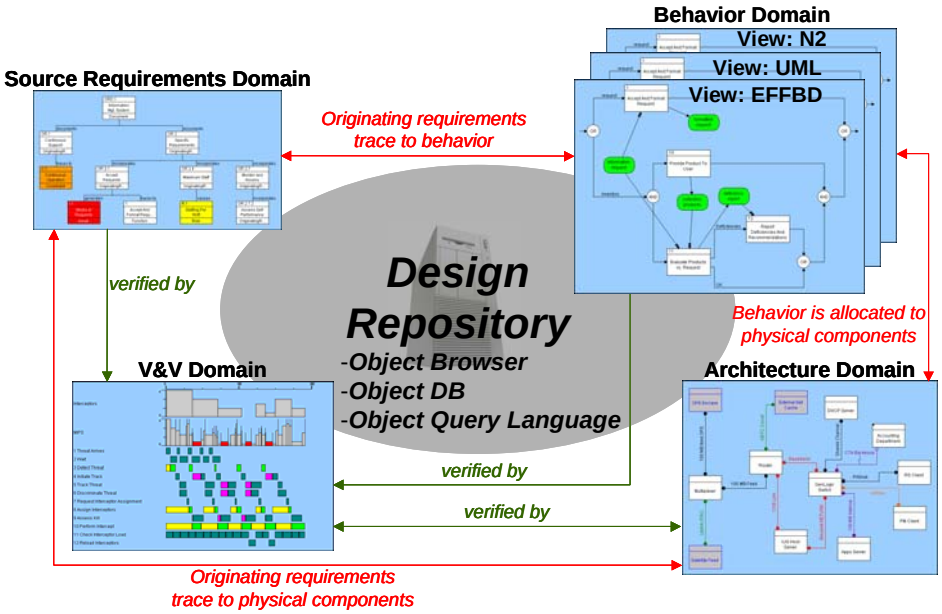
Background

As a practicing systems engineer with extensive background in both software systems development and early UML development, I would propose that UML is "another systems engineering view". However, this view in itself is incomplete and this paper will identify some key issues of why practicing systems engineers will continue using behavior diagrams as a powerful ally in designing and specifying systems.

The executable modeling language used by Vitech Corporation's CORE® tool has provided systems engineers with the ability to construct executable behavior models and assist in many other systems engineering tasks for more than 25 years. These other systems engineering tasks include concept definition, executable scenarios development, requirements analysis, security analysis, allocation of functions to components, and performing detailed integrity analysis. As required, documents and specifications are generated directly and automatically from these models. An Object Query Language (scripts) provides the dynamic execution and reporting capabilities.

Overview of CORE®

CORE provides the systems engineer with the capability to construct executable models using a graphical language to produce several "views." The various views of a system are generated from the Design Repository, based on an Object Database, which maintains integrity of these views. View consistency is provided by the underlying semantic model contained within the CORE repository. For a complete description of view projections from the repository, see Jim Long's paper, "Relationships Between Common Graphical Representations in Systems Engineering" (July 2002) available at www.vitechcorp.com.



Adaptations of CORE to Support UML

What systems engineers and software engineers have not had is a universal standard language and commonly agreed semantics with an appropriate exchange mechanism to support a diverse team that includes: systems engineers, management, software engineers, mechanical engineers, electronics engineers, communications engineers, and text for document production.

CORE, in the support of systems engineering uses a semantically rich language with graphical representations as part of that language. This language has been refined over years of successful application on complex programs.

The CORE systems engineering language can be extended by using basic schema extensions so that projects can accommodate Software Packages, Classes and Methods. Functions from executable scenarios are then "allocated to" methods, bridging the gap between systems engineering and software engineering. Note that the creation of object classes and methods from scenarios is a return to classical object theory. (Object-Oriented Software Engineering: A Use Case Driven Approach, Ivar Jacobson). Timelines approximate UML Activity Models and Sequence Diagrams. Further adaptations can be made to support the UML 2.0 standard as it matures.

Software engineers have not broadly accepted the systems engineering language and semantics because of their constrained solution space.

UML as an Additional Systems Engineering View

UML is the language most adopted by Software Engineers. UML (UML 2.0) is now sufficiently mature to include dynamic execution capabilities. The Systems Engineering profession is moving towards including UML as "another representation" because:

1. Most modern programs have a significant software intensive component.
2. Collaboration between systems engineers and software engineers throughout the program is required.
3. Systems Engineers want to improve communications among stakeholders by using a coherent, consistent language and the executable models expressing the design. Systems Engineers have considerable experience (25 years plus) in building and using executable behavior models while UML 2.0 is an emerging specification and has a limited experience base.

It is in the best interest of the systems engineering profession to include UML where possible, but only if it is done without compromising the consistency, integrity and semantics of behavior models. This guarantees the completeness of system definition, the executability of the integrated design, and the "balance" of the design among the hardware, software, and human components.

A Systems Engineer's Position on UML

To achieve the goal of improved communications between Systems Engineers and Software Engineers, UML can provide "another view" with the following stipulations:

Model Integrity. System Engineers must maintain absolute integrity of the underlying semantic object models necessary to support executable models. All of the underlying semantics must be capable of consistently being "mapped" between behavior models and UML.

Executable Models. Software Engineers are currently using Vitech's CORE behavior modeling capabilities as the standard to develop the UML 2.0 specifications. Element names and constructs may differ (example: and's and or's versus forks and joins) however the terminology mappings are compatible.

Systems Engineering Support of All Disciplines. Systems Engineers will continue developing models and specifications for all disciplines, including hardware, software, peopleware, and communications.

These models and specifications include requirements, functions, components, interfaces, budgets (weight, power) and verification requirements.

Systems Engineers Design of Integrated Control. Systems Engineers are specifically charged with integrating control systems into the behavior of the systems they design. Special conditions, such as periods of high throughput, outages, operating under degraded conditions and returning to normal operations are the specific domains of the systems engineer. The capability to incorporate control systems into the models must be maintained.

Specific Recommendations/Positions

Behavior modeling possesses several capabilities that UML must consider if it will have a sustaining role in systems engineering. Specific examples are as follows

Software Specific Element Support. CORE is currently capable of extending the base component schema to include specific software elements. Design Packages are then passed to all component engineers. Software specifications include Software Deployment Packages, Classes, and Methods. Requirements are "traced to" functions. Functions within executable scenarios are then "allocated to" methods in the component hierarchy. Other elements are related to the components, such as: constraints, performance indices, issues, and risks, and test (V & V) threads.

Mapping of Behavior to UML. The process of mapping the semantics of behavior models to UML diagrams is provided in the following table:

Behavior Models:	UML Diagrams
Functions (application)	Methods
Control constructs (AND, OR, . .)	Mapping UML 2.0 (forks and joins)
Function (multiple exits)	No direct mapping
Loops, Loop Exits, Iterations	Mapping UML 2.0
Function Scripts (random numbers, etc)	No direct mapping
Resources	No direct mapping

System Engineers have come to rely on the rich semantics of behavior models. The primary question: what do we do with the semantics – specifically those that represent controls – that do not map directly to UML?

System scenarios are not equal to UML Use Cases. CORE scenarios are initially developed as threads of behavior. Threads are then aggregated to form operational scenarios. These operational scenarios, in the form of process models, are then aggregated and transformed into product models. These behaviors are then "allocated to" a specific component implementation.

The definition of Use Cases has changed over the last five years. In the past (Jacobson), use cases were defined as detailed text descriptions of the sequence of activities required to describe a "thread" of behavior. Currently, UML defines use cases as I/O's that crosses the systems boundary and, by definition, UML use cases are not capable of decomposition. The point to be made is that decomposable scenarios are required to describe a comprehensive set of behavior. Therefore, executable and decomposable scenarios are required to develop good class / object models. (Jacobson).

Behavior First versus Components First. CORE will support various forms of design methodologies: top down, bottom up (reverse engineering), and middle out. When performing top down design, software classes and methods should be derived from comprehensive systems level scenarios (Jacobson).

Some software engineers want to develop the component hierarchy first (including classes and methods) without performing requirements analysis, constructing executable scenarios, or building product models. Requirements are immediately "associated" with high-level components (software, hardware, electronics).

No systems engineering is performed, therefore, there are no systems level behavior models, allocation to components, interface definitions, budgets, or verification requirements to be met at integration and test. This is not a problem with UML as a language, but with the application of UML.

Exclusions of Humans from the Models. UML defines use cases as I/O's that cross the systems boundaries. Human activity is generally left outside the systems boundary. These use cases are not particularly useful in describing the interaction between a human and the system. Comprehensive scenarios representing interactions between the human and the system are required.

Object Query Language. CORE currently has a comprehensive Object Query Language. The Object Query Language is required to (a) navigate the collection of elements, (b) support detailed executable programs (CORE scripts) embedded within the models, and (c) produce text specifications from the models. UML does not specify nor preclude such a language.

Conclusions

Systems Engineers have a rich language with semantics and representations based upon years of practice in supporting executable models and specifications. The underlying integrity and semantics of the models are much more important than the graphics representations of specific UML diagramming techniques.

There is a need to provide congruent views to both the systems engineering and software engineering communities. UML representations should and will be supported as "another view" of the underlying models as the UML 2.0 specifications continue to mature.

UML representations should and will be supported and accommodated under the conditions that:

1. The integrity of the underlying semantic models will not be compromised, and,
2. The contents of the underlying semantic models essential to complete definition of system components and logic – including automated dynamic validation of behavior – can be mapped to UML views.

Unless these two conditions are met, then System Engineers will be required to continue to use behavior models in their present form so they can provide complete, consistent, executable, validated system-level designs to their subsequent subsystem developers.

About the Author

J. F. Skipper, Ph.D. is a Senior Systems Engineer for Vitech Corporation. He is currently engaged with the AP-233 Standards initiative sponsored by JPL/NASA and is actively participating in the integration of Systems Engineering and Software Engineering processes as a member of the SysML team customizing UML 2.0 to support systems engineering applications (www.sysml.org).

His previous twenty years of professional experiences includes BEA WebLogics application servers and development of a JAVA run time engine. As a Senior Consultant, Object Technology Practice, IBM Corporation, he provided object solutions for distributed (WebSphere) customers. Previous activities focused upon creating discrete event simulation models, producing detailed specifications for software programming teams using UML in their software development process.

Updated June 2003