Systems engineering in the product lifecycle

Conrad Bock

US National Institute of Standards and Technology,
Manufacturing Engineering Laboratory,
100 Bureau Drive, Stop 8263,
Gaithersburg, MD 20899 8263, USA
Fax: 301 975 8273 E-mail: conrad.bock@nist.gov

Abstract: This paper introduces basic elements of systems engineering that are useful in managing the product lifecycle, as expressed in an extension to the Unified Modeling Language. It presents models of product requirements for capturing stakeholder needs, system structure for defining the static relations of its elements, behaviour for the transformation of inputs to outputs, parametrics for constraining properties of structure, and allocation for assigning behaviour to structure. The relation of behaviour to structure is identified as a central issue in the integration of systems and software engineering.

Keywords: product lifecycle; systems engineering; UML; SysML; product modelling.

Reference to this paper should be made as follows: Bock, C. (2005) 'Systems engineering in the product lifecycle', *Int. J. Product Development*, Vol. 2, Nos. 1/2, pp.123–137.

Biographical notes: Conrad Bock is a Computer Scientist at the US National Institute of Standards and Technology specialising in product modelling and UML. He leads efforts on UML process modelling at the OMG, and is contributing to the submission on OMG's UML for Systems Engineering. Bock studied at Stanford, receiving a BS in Physics and a MS in Computer Science. His previous experience includes expert systems in vehicle design and nuclear power.

Systems engineering (SE) overlaps a significant portion of product lifecycle management (PLM). It has existed as a discipline for several decades and been applied successfully to a wide range of complex products (INCOSE, 2000). This paper introduces SE and recent work on defining a standard SE modelling language. It also addresses a central aspect of integrating software development into general engineering practice.

1 Introduction

Both PLM and SE are concerned with managing the multiple views and interrelationships of product information to maintain coherence across time and place, and to apply to more than one product. The International Council on Systems Engineering (INCOSE) describes SE as:

> "…defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem: operations, performance, test, manufacturing, cost and schedule, training and support and disposal." (INCOSE, 2004)

The areas addressed by SE are so diverse that it must be concerned with communication between people working in them:

> "Systems engineering integrates all the disciplines and specialty groups into a team effort forming a structured development process that proceeds from concept to production to operation." (INCOSE, 2004)

SE is currently hampered by a lack of a standard language for coordination across the product lifecycle and across disciplines involved in product development. Organisations using multiple languages have less effective communication, increased project cost, and decreased product quality. Many of the specialties that systems engineering interacts with have adopted standard languages, most recently software engineering.

To address these issues, INCOSE joined with a major software consortium, the Object Management Group (OMG), to create a Standard Modelling Language for Systems Engineering. OMG and INCOSE began by forming the Systems Engineering Domain Special Interest Group (SE-DSIG) (Friedenthal and Kobryn, 2004; OMG 2004a). The SE-DSIG developed a request for proposal for an SE modelling language, issued in March 2003 (OMG, 2003a). The requirements were developed from an OMG request for information (OMG, 2002) and with INCOSE and the international organisation for standardisation's ISO 10303, informally known as the standard for the exchange of product model data (STEP). In particular, ISO's 10303-233 application protocol for systems engineering team (AP-233) participated to align the OMG requirements with the evolving AP-233 neutral data interchange standard for systems engineering (ISO, 2004a). This is important in bridging to engineering analysis disciplines represented in other ISO standards.

The OMG request covers a substantial part of SE:

- requirements
- structure
- behaviour
- parametrics (constraints)
- verification (testing)
- deployment.

The request also identifies OMG's Unified Modeling Language (UML) (OMG, 2004c) as a basis for SE modelling, because it combines critical elements needed for SE:

- graphical presentation for communication between a wide variety of disciplines
- extensibility mechanisms for adapting to new domains
- wide range of capabilities, from requirements to deployment
- model repository supporting notations for multiple disciplines, and translatable to multiple specialised formats.

The last feature addresses the essential requirement in PLM and SE for a common repository capturing consistent information accessible across the lifecycle. It also provides information beyond geometric and mathematical representations commonly used in specifications of product data (Sriram, 1997). The variety of product data requires flexible representations available in languages such as UML, or ontologies such as Ontology Web Language (OWL), and Process Specification Language (PSL) (Staab and Studer, 2004). These languages support identification of terms, relations, and processes in a form that is amenable to automated reasoning. In the setting of OMG's model-driven architecture (MDA), they can be translated to optimised formats for other applications as necessary, such as automated manufacturing (OMG, 2004b; Bock, 2003c).

This paper describes a response being prepared to the OMG request called the Systems Modeling Language (SysML) (SysML Partners, 2004). The first version will cover requirements, structure, behaviour, parametrics, and the relation of structure to behaviour (allocation). Each of these is introduced in the sections below. Finally, the relation of structure and behaviour is examined as a central issue in the integration of systems and software engineering.

2 Requirements

Requirements modelling includes the translation of textually expressed needs into a computable form, capturing the evolution of requirements and their relation to the system design. There are at least three important aspects to this process:

1 *Translation of text to model.* Requirements usually appear first as a large text document, structured in some way by headings. The requirements model is a 'parsed' form of the document. It has an element for each requirement, containing the portion of text corresponding to that requirement, with a reference to its location in the source document. The model contains a single element per requirement, even when the source text conjoins them.

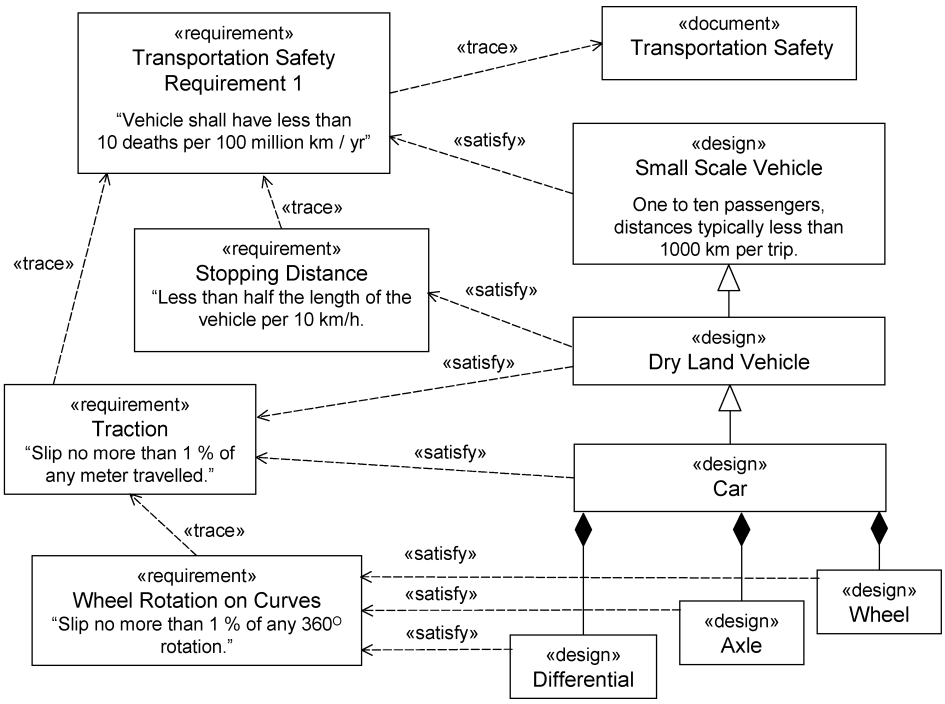
2 *Derivation.* Requirements can be more or less specific about the requested system. For example, a source requirement for transportation might be about safety generally. For automobiles, braking distance, traction requirements, and so on, are derived from safety. These further derive requirements on wheel rotation forces and speeds. SysML calls this a requirements trace. Derivation can include detailed models, as in UML structure and behaviour. Each stage of derivation will involve some assumptions about the design of the system (Cantor, 2003). For example, the derivation of the safety requirement above assumed the transportation mode was an automobile of the conventional kind.

3 *Link to system design.* System designs that fulfil requirements are said to satisfy the requirement. SysML anticipates that system designs will be modelled in UML, but the satisfaction relationship does not restrict how the design is represented.

SysML adds a requirements model to UML, because UML does not have one. Figure 1 shows the transportation safety example, with derivation traces based on increasingly elaborated designs ('requirements flowdown'). The guillemot notation («») in UML is used to indicate what kind of language element a particular rectangle or arrow represents. For example, the rectangle at the top right labelled «document»

TRANSPORTATION SAFETY is an original source requirements document for the system being designed, and «requirement» TRANSPORTATION SAFETY REQUIREMENT 1 is a part of the model representing one of the requirements in the text document. Dashed arrows in UML show dependencies between pieces of the specification, where the element at the arrowhead end is independent and the other end is dependent on it. For example, the dashed arrow labelled «trace» indicates that the requirement is 'parsed' from the original source document, and depends on it.

Figure 1 Requirements



The rectangles marked «design» in Figure 1 represent classes of physical objects, where the design is specified in the class, and describes the structure of the physical objects that are members of the class. Designs are constructed to satisfy requirements, as indicated by the «satisfy» dependencies. Some of the designs are connected by generalisation arrows, notated with a hollow arrowhead. The design at the arrowhead end is a general case of the design at the other end, which inherits characteristics from the general design. For example, the characteristics of physical objects conforming to the SMALL SCALE VEHICLE design also apply to objects conforming to DRY LAND VEHICLE. In particular, small-scale vehicles are designed to satisfy TRANSPORTATION SAFETY REQUIREMENT 1, as shown by the dashed arrow labelled «satisfy», so dry land vehicles will inherit the characteristics satisfying that requirement also.

Each specialised design introduces new characteristics that satisfy requirements derived from the general design requirements, as shown by the «trace» dependency. For example, the additional characteristics introduced by DRY LAND VEHICLE satisfy the TRACTION requirement derived from TRANSPORTATION SAFETY REQUIREMENT 1. This pattern also applies to parts of designs, as shown at the bottom

right of the figure, using UML's black diamond line notation. These lines represent the assembly breakdown of physical objects in the classes being associated. For example, differentials are parts of cars, and satisfy the corresponding derived requirement WHEEL ROTATION ON CURVES, even though they are not special cases of cars.

The designs in this example are structural, but some methodologies use functional design (see Sections 4 and 6). SysML also supports capturing the reason for any particular derivation or design choice, called the *rational*. These can include trade-off analyses. Requirements can specify testing procedures, to ensure the design actually satisfies the requirement. The trace and satisfaction relationships can be grouped, for example to identify alternative design choices. Requirements can also be expressed in tabular formats to improve scaling, which is necessary for typical systems. This technique is based on the repository for UML and its extensions, which records models in a way that is independent whether they are presented diagrammatically or textually (Bock, 2003c).

3 Structure

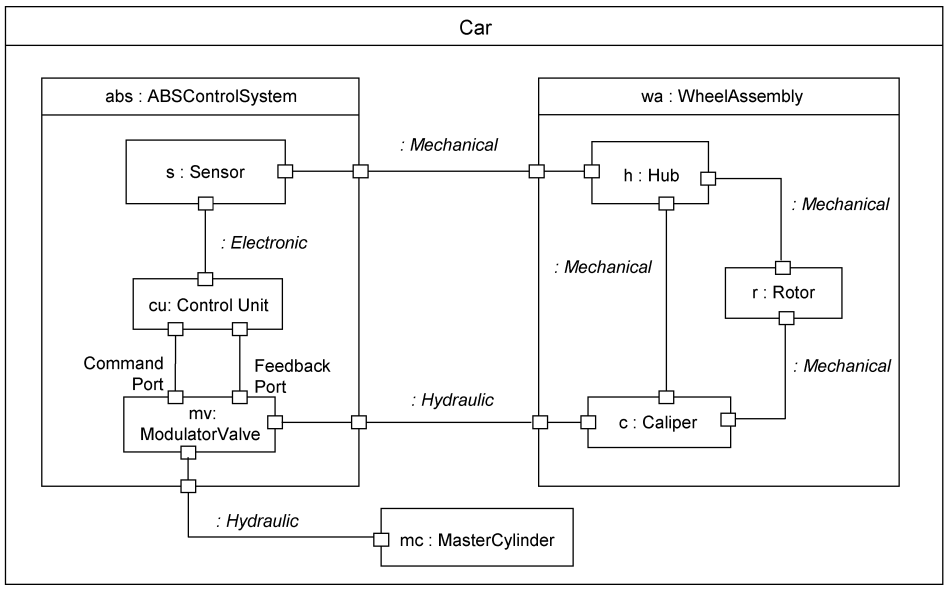
SysML reuses the UML 2 composite structure model, which SysML calls *assemblies*. The UML 2 model adds significant power to the earlier versions of UML, in particular, it supports the reuse of system elements in multiple assemblies, or in multiple ways in the same assembly (Bock, 2004b). Figure 2 shows an example composite structure for a simplified automatic braking system (ABS). The outer rectangle represents the class of cars, members of which are individual physical cars. It carries the design that each car conforms to. The labelled rectangles inside the car are its parts, for ABS, wheel assembly, and so on. Some of the parts are subassemblies, such as the one for wheels. These have their own components and interconnections inside. The small rectangles on the borders are ports, which provide access points for connections outside the part. For example, the ABS has ports for mechanical and hydraulic connections. The lines segments between ports are connectors, which show how the parts are connected together electronically, hydraulically, and so on.

Each component of the car is a usage of a generic, reusable subcomponent. For example, the wheel subassembly might be used four times, though the figure shows only one for simplicity. Each usage of a wheel subassembly will be connected in different ways, for example to different ends of the same axle, or to different axles entirely. To accommodate multiple usages of the same kind of subassembly, the usage of each component is given a name, which appears to the left of the colon at the top of each part. The type of part being reused appears to the right of the colon. For example, the generic WHEELASSEMBLY is used under the name WA. When other wheels are added to the design they will have different usage names, even though they are all usages of WHEELASSEMBLY.

The UML 2 composition model conforms to the common intuitions of assembly. For example, the ABS in one car is not connected to the wheels in another, and the ABS connects only the wheels indicated in the design, which might not be all the wheels in the car, and some cars might have wheels without ABS systems. The UML class diagram does not support these intuitions, because class diagrams define artefacts generically, independently of how they are used. For example, a class diagram showing an association between ABS systems and wheels would either need to define the linkage

as optional, which would be incorrect for cars that have ABS, or make the linkage required, which would be incorrect for cars that do not have ABS. The class diagram must be used in conjunction with the composite structure diagram to show how generic, reusable components are applied in particular designs (Bock, 2004b; Baysal et al., 2004).[1]

Figure 2 Composite structure



4 Behaviour

One of the purposes of behaviour models is to coordinate or place constraints on other behaviours.[2] For example, a procedure for shaping a piece of metal might have a series of steps that must happen in a certain order under certain conditions. UML provides three behaviour models that are reused in SysML. Each kind emphasises a different aspect of system dynamics, making one or the other more suitable for a particular application, or stage of application development:

- *activities* emphasise inputs and outputs, conditions, and sequence for invoking other behaviours
- *state machines* show how events cause changes of object state and invoke other behaviours
- *interactions* describe message passing between objects that cause invocation of other behaviours.

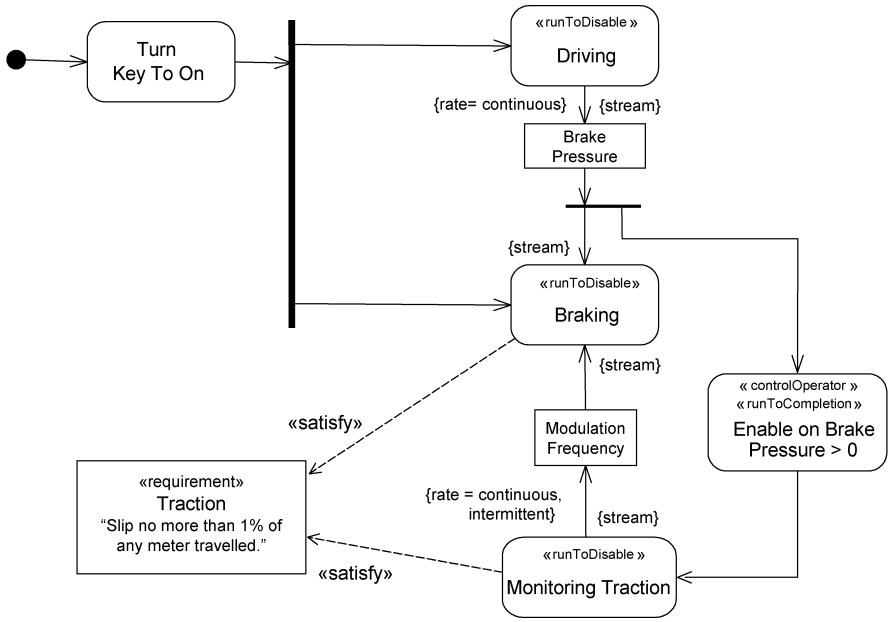
Activities are of particular interest to SE because they focus on what tasks need to be done, in what order, with what inputs, rather than which entity performs each task (Bock, 1999, 2003b). More specifically, activities are designed to be used with or without objects (see Section 6), and highlight the dependency of inputs on outputs. This emphasis

corresponds to SE functional flow, in particular Enhanced Functional Flow Block Diagrams (EFFBD) (Bock, 2003a). Functional flow is closest to requirements expressed as transformations of inputs to outputs (also called *functions*).

Activities follow the style of UML 2 composition models in supporting multiple uses of the same behaviour. For example, system requirements might dictate a subfunction for heating water that takes water as input and provides it as output at a higher temperature. This generic, reusable function might be applied in many ways, and in each particular application it might provide hot water to different downstream functions. For example, one usage in a ship might provide hot water to a room heating function, another to a cooking function, and so on.

Figure 3 is an example activity diagram, with some SysML extensions, showing the dependencies between some subfunctions of an automobile, adapted from SysML Partners (2004). Round cornered rectangles in activity diagrams represent the usages of functions, which are called *invocations*. For example, the one on the upper left is an invocation of a generic, reusable function that turns the key to the on position. Arrows, vertical bars, and dots in activity diagrams determine when the function invocations occur. For example, the dot on the upper left is an initial node, representing the starting point of the activity. The arrow coming out of it is a control flow, indicating the first step in the activity, TURN KEY TO ON.

Figure 3 Activity example 1



Vertical and horizontal bars in Figure 3 are forks showing initiation of concurrent flows of function invocations. For example, after the key is turned on, driving and braking functions start concurrently, because the control flow coming out of TURN KEY TO ON is split into two concurrent control flows, to DRIVING and BRAKING. The SysML keyword «runToDisable» on a function invocation indicates that once the function starts, it runs until it is turned off (turning the key to off is not shown).

The arrow coming out of DRIVING is an object flow, also called a data flow, or *item flow*. It represents the flow of information, material, or energy between functions. Item flow is distinguished from control flow by a rectangle indicating what type of thing is flowing. In this example, brake pressure information is passing from driving to braking. The information passed between functions is not necessarily implemented electronically or in software. For example, sending brake pressure information can be implemented in an analogue fashion, as with conventional hydraulics. The STREAM properties in curly braces indicate that this information might be passed between the functions while they are operating, rather than requiring DRIVING to complete before generating an output, or BRAKING to wait for break pressure to arrive before starting. The SysML RATE properties specify how fast the information flows. In this example, break pressure is emitted continuously from the driving function.

The horizontal bar below BRAKE PRESSURE in Figure 3 indicates that the information is sent concurrently to two function invocations, one for BRAKING and the other for a special function that controls MONITORING TRACTION. This function is a control operator, which means it outputs control information that can enable or disable other functions (Pandikow and Torne, 2001). When the input break pressure is greater than zero, it outputs an enabling control value to monitor traction, otherwise it emits a disabling control value. The effect is that traction is only monitored when pressure is applied to the brake. The SysML keyword «runToCompletion» indicates the control operator starts when it receives an input, calculates its output, then stops until the next input arrives. It is not intended to run indefinitely like driving and braking, though it happens to run repeatedly in this example, because its inputs come in continuously (also see example in Figure 4). While MONITORING TRACTION is running, it emits modulation frequencies to BRAKING as necessary to maintain traction.

Figure 4 Activity example 2

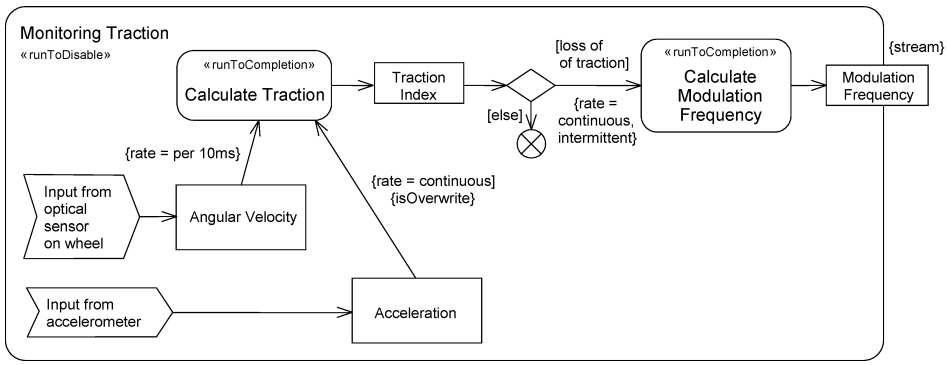


Figure 4 shows the definition for the MONITORING TRACTION function used in Figure 3. The chevron shapes on the lower left represent receipt of signals from outside the process, which do not come through explicitly represented inputs. When enabled, MONITORING TRACTION receives signals from the wheel and accelerometer, which flow to a subfunction that calculates traction. This function is marked as «runToCompletion», so it will wait for two inputs to arrive, calculate the traction index output, stop, and wait for two more inputs. The acceleration input comes in faster than the angular velocity, as shown by the RATE properties, causing acceleration data to queue up

at CALCULATE TRACTION while it waits for an angular velocity to pair with. To prevent a stale acceleration value from being paired with a new angular velocity, the acceleration input is marked with the SysML IS OVERWRITE property, which causes new values of acceleration to overwrite old ones in the queue to CALCULATE TRACTION.

The diamond shape in Figure 4 is a decision node that routes the output of CALCULATE TRACTION according to guard conditions notated in square brackets on the arrows coming out of the decision. One of the guards tests whether enough traction is lost to justify outputting a modulation frequency for the brakes. If so, the traction index flows as input to a function that calculates the frequency, the output of which passed out of the function. If not, the ELSE guard directs the values to a flow construct that discards it, notated as a circle with an X in it.

The UML repository can support tabular or matrix formats such as the dependency structure matrix (DSM) (Sharman and Yassine, 2004). These provide a compact way to show function dependencies, by omitting some control information, but are not restricted to hierarchical decomposition. A DSM can be derived from activity models, stored in a repository, analysed, and results presented in either a matrix notation or an activity diagram.

Functions can satisfy requirements, as shown in the lower left of Figure 3.[3] Under some methodologies system function is determined separately from structural design, and requirements are satisfied through function, which is then allocated to structure, rather than satisfying requirements by structure directly, as in Figure 1 (see Section 6) (USDoDSMC, 2001).

5 Parametrics

A parametric model describes constraints among properties of a system. These are typically expressed as mathematical equations, for example 'F = Ma'. UML provides a constraint language (OMG, 2003b), but it does not currently support for reusing equations, a critical requirement for SysML. For example, 'F = Ma' can be one of a library of equations that are reused many times in the analysis of a system, with variables bound to different properties in each case. SysML introduces a constraint model supporting reusable equations, which is called *parametric relations*.

The SysML parametric model has two parts:[4]

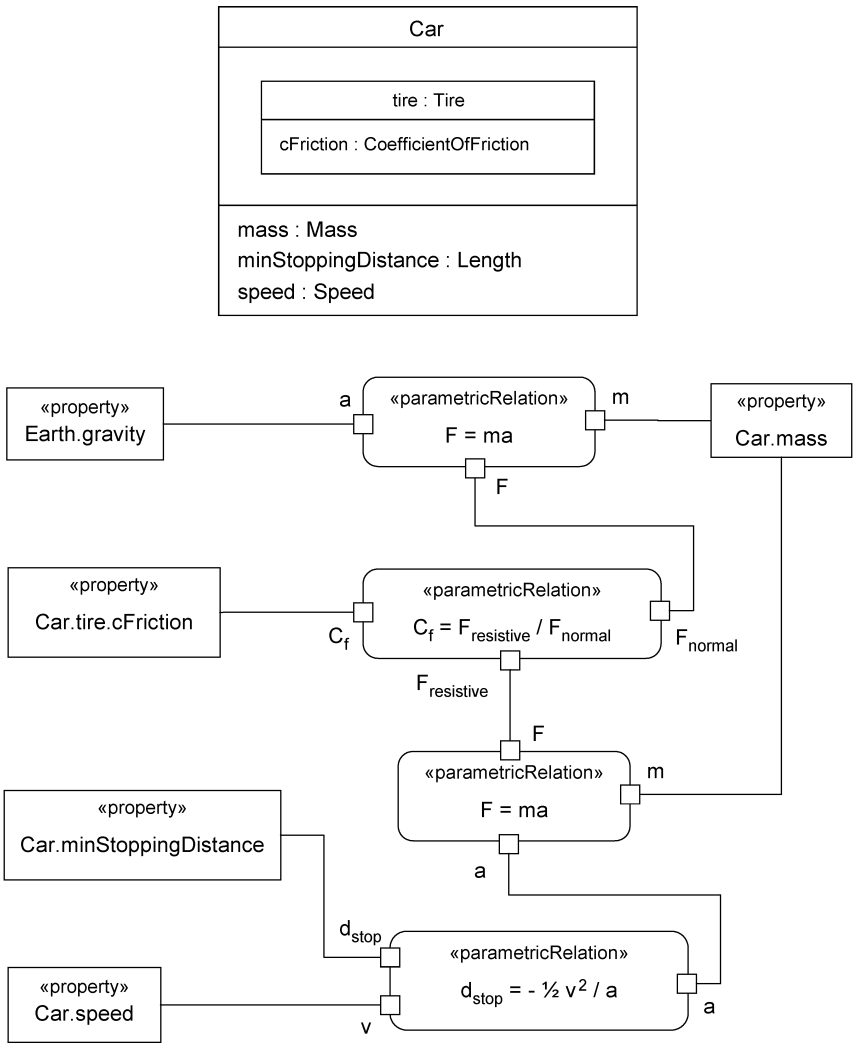
- Parametric relations defined as a reusable combination of other relations.
  For example, 'F = Ma' is a combination of equality and multiplication, both of which are primitive parametric relations.
- Application of a parametric relation to particular object properties.

The lower part of Figure 5 shows a SysML parametric diagram relating stopping distance to other parameters in an automobile, along with a partial structure model that it refers to at the top. The rectangles with keyword «property» represent the properties being constrained. The rounded rectangles represent parametric equations. The small rectangles on the borders are the parameters being constrained, which are connected to each other and to properties, forming a network of constraints. The equation 'F = Ma' is used twice,[5] connected to different relations and properties in each case, but defined once in the repository that stores the diagram (Bock, 2003c). This means that any change

to the equation is propagated to all uses in all diagrams (not that one would ever change Newton's laws).[6]

Parametrics are intentionally nondirectional, that is, they do not have inputs and outputs as activities do. For example, the equation 'F = Ma' constrains three variables, but it does not specify which are being calculated. This applies even when using irreversible relations, as in 'y = sin(x)'. The model leaves it up to constraint engines implementing the equations to determine what to do in these cases. For example, in Figure 5, the coefficient of friction of the tyre could be a dependent or independent variable, depending on whether it is given as an input to the constraint engine.

Figure 5 SysML parametrics example



For compactness, parametrics can be notated as equations, and still be stored to the SysML repository. Tools can use the repository to generate the equations from diagrams such as Figure 5, or vice versa. The SysML repository does not dictate or suggest

a mathematical notation for parametrics, only a way to identify relations and how they are used.

Parametric models can apply to any kind of value. They can be physical properties of the system, performance and effectiveness measures to choose between alternative designs, engineering analysis values, or any combination. The special property time is treated as the property of a global clock.

6 Allocation

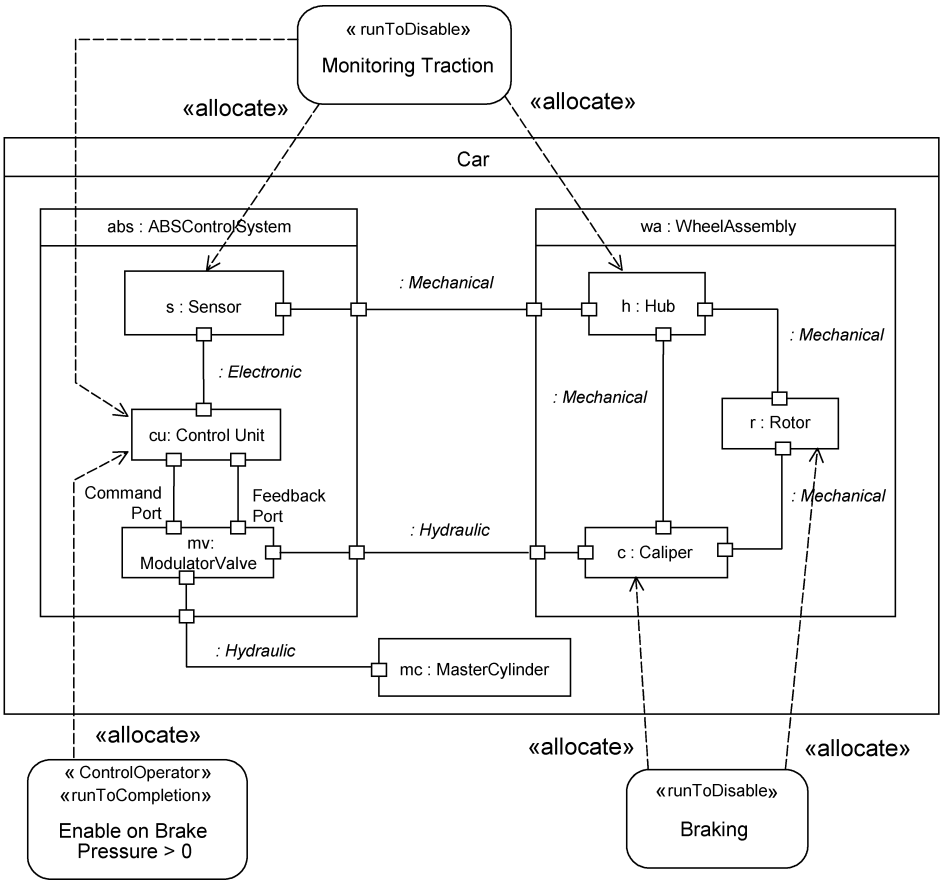
Behaviour and structure must be used together to produce a real system, and SE takes the relation of behaviour to structure as a special area of concern, called *allocation*. SE requires a flexible connection between behaviour and structure because many alternative structures can support a required behaviour or function. Even when a structure is chosen, behaviour is often spread across multiple structural elements. For example, an automobile's braking system has the overall function of translating driver input into change in the speed of the car, but this might be achieved through many structures, for example with hydraulics and mechanical pressure, or with electronics and electro-mechanical devices, and so on. Within any one of these structural choices, the function is achieved by multiple structural elements, like the hub and a portion of the axle in the wheel, the hydraulic system attached to the frame, and elements in the passenger compartment.

Even in situations where structural elements are determined as part of requirements, for example in systems based on common off the shelf (COTS) components, it is beneficial to capture the existing functionality separately from structure. COTS products are never completely uniform in structure, have a variety of unintended effects, and are available in many versions developed over time.[7] In addition, the new functionality arises from the synergy of existing components, not the components themselves, and there might be many possible ways of assembling them. These factors make it simpler to address the desired functionality separately from how it is achieved by the mandated COTS components.

Another advantage of addressing behaviour independently of structure is that behaviour models can be simulated in the context of model-driven architectures, for testing early in the development cycle. Then dependencies between functions can be verified for all the alternative structures that might support them. This includes dependencies between inputs and outputs and between postconditions and preconditions. Complex systems have many interdependent functions with overall behaviour that is hard to predict. Simulation helps to evaluate the total behaviour for all the alternative structures at once, and determine the theoretical boundaries for performance of those structures.

SysML provides a dependency for allocating behaviour to the structure supporting it. For example, Figure 6 shows the allocation of the braking, monitoring, and enabling functions to parts of the structural model. For example, the monitoring traction function is performed by the combination of sensor and hub. Allocation can also relate the flows in an activity model to the connectors in the structural model. For example, the modulator frequency that is passed from MONITORING TRACTION to BRAKING in Figure 3 could be allocated to the hydraulic connector between the ABS and wheel assembly in Figure 6.

Figure 6 Allocation



As usual in model-driven approaches, other notations for allocation can be supported, for example to show functions inside the part rectangles, or in a tabular format, and so on. UML also supports a notation that groups the steps in an activity, called *partitions*, which can be also used for allocation (Bock, 2004a).

7 Integration of systems and software engineering

Software is increasingly important in SE and PLM, because it controls larger portions of modern systems, causing a wider range of errors (Kasser, 2000). In addition, software engineering is not well integrated with other disciplines, because it is more recent. For example, ISO assigns software standards and product information interoperability under separate technical committees (ISO, 2004b).[8]

One of the central issues arising in the integration of systems and software engineering is the different approaches they take to the relation of structure and behaviour. Systems engineering and most specialised disciplines allow a flexible connection between them, as described in Section 6, whereas modern software development methodologies usually keep them tightly bound, for example in object orientation.[9] Specifically, these approaches select a single object to perform each subfunction,[10] then define the interaction between objects to perform the overall function. Changing the object that supports a subfunction requires moving the service from one object to another, and changing the interaction between them. It forces the modeller to manage two difficult problems at once: dependency between functions and interaction between objects.

Fortunately, UML 1.5 and 2.0 reintroduced the flexible relation of behaviour and object that were present in early software methodologies, while maintaining the option of tight binding between them. For example, the behaviour in Figure 3 only loosely constrains the structure that supports its subfunctions, which could be Figure 2 or some other structure, and once the structure is determined, it only partially constrains how the subfunctions are allocated, which might be as in Figure 6 or some other allocation. Figure 3 only declares the dependency of subfunctions on each other, through inputs, outputs, and control.

At the same time, UML 2 activities support the tighter binding of behaviour and structure found in modern software methodologies. In these approaches, each subfunction is supported by a single object that receives requests to perform that function. The transition from loose to tight binding only requires local changes, so a model embodying the SE approach to behaviour and structure can be easily transformed to an object-oriented one.

8 Conclusion

This paper reviews some of the major aspects of SE, which cover a substantial portion of PLM: requirements, structure, behaviour, and parametrics. These are supported by work on a standard SE modelling language, SysML. SysML follows general SE practice by providing model-based requirements for precisely capturing stakeholder needs, including requirement derivation and link to design solutions, structure modelling for intuitive description of assemblies, behaviour modelling for functions and function dependency, and parametrics for reusable calculations and equations. The relation of behaviour to structure in SE is more flexible than modern software approaches, and SysML utilises recent developments in UML to integrate these. It is hoped that SysML will be a significant contribution to model-based PLM in general.

Acknowledgements

Thanks to Sanford Friedenthal for his input to this paper and leadership in the UML for Systems Engineering effort.

References

Baysal, M., Roy, U., Sudarsan, R., Sriram, R.D. and Lyons, K. (2004) 'The open assembly model for the exchange of assembly and tolerance information: overview and example', *Proceedings of the 2004 ASME International Design Engineering Technical Conferences & Computers and Information In Engineering Conference DETC2004/CIE'04*, September 28–October 2, Salt Lake City, Utah.

Bock, C. (1999) 'Three kinds of behavior model', *Journal of Object-Oriented Programming*, Vol. 12, No. 4, July–August, pp.36–39.

Bock, C. (2003a) 'UML 2 activity model support for systems engineering functional flow diagrams', *Journal of the International Council on Systems Engineering*, November Vol. 6, No. 4, pp.249–265.

Bock, C. (2003b) 'UML 2 activity and action models', *Journal of Object Technology*, July–August, Vol. 2, No. 4, pp.43–53, http://www.jot.fm/issues/issue_2003_07/column3.

Bock, C. (2003c) 'UML without pictures', *IEEE Software Special issue on Model-driven Development*, September–October, pp.33–35.

Bock, C. (2004a) 'UML 2 activity and action models, part 5: partitions', *Journal of Object Technology*, July–August, Vol. 3, No. 7, pp.37–56, http://www.jot.fm/issues/issue_2004_07/column4.

Bock, C. (2004b) 'UML 2 composition model', *Journal of Object Technology*, November–December, Vol. 3, No. 10. pp.47–73.

Cantor, M. (2003) 'Rational unified process for systems engineering: part III: requirements analysis and design', *Rational Edge*, http://www-106.ibm.com/developerworks/rational/library/content/RationalEdge/oct03/m_rupse_mc.pdf, October.

Denno, P., Steves, M., Libes, D. and Barkmeyer, E. (2003) 'Model-driven integration using existing models', *IEEE Software Special issue on Model-driven Development*, September–October, pp.59–63.

Fenves, S. (2002) 'A core product model for representing design information', *US National Institute of Standards and Technology Internal Report 6736*, October.

Friedenthal, S. and Kobryn, C. (2004) 'Extending UML to support a systems modeling language', *INCOSE Symposium Proceedings*, Toulouse, France.

INCOSE (2000) S*ystems Engineering Applications Profiles*, http://www.incose.org/ProductsPubs/pdf/techdata/SEApps-TC/SEApplicationsProfiles_2000-07.pdf, July.

INCOSE, (2004) *What is Systems Engineering*, http://www.incose.org/practice/whatissystemseng.aspx.

ISO, International Organization for Standardization (2004a) TC184/SC4/WG12 10303-233, *STEP Application Protocol for Systems Engineering*, http://step.jpl.nasa.gov/AP233/.

ISO, International Standards Organization (2004b) *List of Technical Committees*, http://www.iso.org/iso/en/stdsdevelopment/tc/tclist/TechnicalCommitteeList.TechnicalCommitteeList.

Kasser, J. (2000) 'Systems engineers are from Mars, software engineers are from Venus', *Proceedings of the Thirteen Annual International Conference on Software & Systems Engineering and their Applications*, December.

Kiczales, G., des Rivieres, J. and Bobrow, D. (1991) *The Art of the Metaobject Protocol*, MIT Press.

Martin, J. and Odell, J. (1992) *Object-oriented Analysis and Design*, Prentice-Hall, Englewood Cliffs, NJ.

OMG (2002) 'UML for Systems Engineering RFI', *Object Management Group Systems Engineering Domain Special Interest Group*, http://www.omg.org/cgi-bin/doc?ad/02-01-17, January.

OMG (2003a) 'UML for systems engineering RFP', Object Management *Group Systems Engineering Domain Special Interest Group*, http://www.omg.org/cgi-bin/doc?ad/03-03-41, March.

OMG (2003b) 'UML 2.0 OCL specification', *Object Management Group*, http://www.omg.org/cgi-bin/doc?ptc/03-10-14, October.

OMG, (2004a) *Object Management Group Systems Engineering Domain Special Interest Group*, http://syseng.omg.org/.

OMG (2004b) 'Model-driven architecture', *Object Management Group*, http://www.omg.org/mda/.

OMG (2004c) 'UML 2.0 superstructure specification', *Object Management Group*, http://www.omg.org/cgi-bin/doc?ptc/04-10-02, October

Pandikow, A. and Törne, A. (2001) 'Integrating modern software engineering and systems engineering specification techniques', *Proceedings of the Fourteenth Annual International Conference on Software & Systems Engineering Applications*, Paris, December.

Sharman, D. and Yassine, A. (2004) 'Characterizing complex product architectures', *Journal of the International Council on Systems Engineering*, Vol. 7, No. 1, pp.35–60.

Sriram, R. (1997) *Intelligent Systems for Engineering: A Knowledge-Based Approach*, Springer-Verlag.

Staab, S. and Studer, R. (2004) (Eds.) *Handbook on Ontologies*, Springer Verlag, January.

SysML Partners, (2004) *Systems Modeling Language: SysML*, http://www.omg.org/cgi-bin/doc?ad/04-08-03, August.

USDoDSMC, US Department of Defense Systems Management College (2001) *Systems Engineering Fundamentals*, http://www.dau.mil/pubs/pdf/SEFGuide 01-01.pdf, January.

Notes

[1]The UML 2 composition model is extensible for manufacturing-specific information, such as abstract descriptions of mechanical connections, kinematics, and tolerances that support modelling early design stage product decisions (Baysal *et al*., 2004).

[2]This is the UML and SysML meaning for the term 'behaviour.' Other models use it to mean the result of applying dynamic physical laws to structure (Fenves, 2002).

[3]Dependency groups could be applied here to indicate that the two satisfaction relations work together to fulfil the requirement.

[4]A third part being considered is to use parametric relations to constrain the values of inputs and outputs of behaviours.

[5]As opposed to using energy-based relations, which are simpler.

[6]A complete version that works with the Mars lander, for example, would model Earth as an instance of a planet class and be part of the total structure model.

[7]This is analogous to reverse engineering in software, which can be extended to ontology extraction (Denno *et al*., 2003).

[8]ISO/IEC/JTC 1 and TC 184 (ISO, 2004b).

[9]A notable exception is James Odell's Object-oriented Information Engineering (Martin and Odell, 1992).

[10]Some approaches can have a single function supported by multiple objects (Kiczales, 1991).