# Building Bridges Between Systems and Software with SysML and UML

Matthew Hause
ARTiSAN Software Tools
Eagle Tower, Suite 701
Cheltenham, Gloucestershire,
GL50 1TA
United Kingdom
Matthew.Hause@artisansw.com

Francis Thom
ARTiSAN Software Tools
Eagle Tower, Suite 701
Cheltenham, Gloucestershire,
GL50 1TA
United Kingdom
Francis.Thom@artisansw.com



**Abstract.** Systems are becoming increasingly reliant on software. One of the roles of the systems engineer is to perform a trade-off analysis of the different architectural solutions to a problem, and allocate requirements to different engineering domains within that solution, including software. It is important to investigate effective ways of establishing traceability from the system definition to the software and other requirements. The Systems Engineering Language, (SysML), which is based on the Unified Modeling language (UML), is being increasingly used by systems engineers to model systems. As well as providing system requirements, SysML models can be used to define the system architecture to be used by the software engineers. In this paper, we will demonstrate how SysML and UML can effectively work together to provide an effective handover between systems and software.

## Introduction

The phrase "Essentially, all models are wrong, but some are useful" has been attributed to Professor George E. P. Box, (Box, Draper, 1987). This is appropriate because modeling by its very nature creates an incomplete replica of the problem or system. Modeling systems is based on abstracting the characteristics of a domain of interest (e.g. the *problem* or the *solution*) to the modeler and ignoring the others. In computer science, abstraction is the mechanism and practice of factoring out *details* so that one can focus on a few concepts at a time (Illingworth, et al, 1991). Depending on the level of abstraction, and the focus of concern, there will be different viewpoints that can be applied to the system. In addition, each level, and each viewpoint will have its own unique elements. An example familiar to most will be internet mapping systems such as Google™ Earth or Mapquest. At the highest level, complete countries are visible with only the major roads available. As the zoom level is increased, addition roads and information become visible until individual buildings can be seen. For SysML, some viewpoints will provide information to be allocated to several engineering domains (e.g. hardware, software, procedural or mechanical etc.). For modeling centered on a particular domain, modeling will involve “abstracting the abstraction” to take into account the information that is appropriate to the viewpoint and the domain. Additional information, application of standards, constraints, etc, will then be added to the model. In other words, systems, software and hardware engineers will all look at a problem from their own points of view.

**The Development Lifecycle.** From the start of the program, there is always a "product" to deliver to the customer that takes on different, increasingly detailed forms, over time and as work gets approved. The baseline is established at a milestone event, and typically given these names:

- "as required" (aka Functional baseline) - the requirements specification.
- "as designed" (aka Allocated baseline) - the design specification, which might be high, and later, detailed design.
- "build-to" (aka the development baseline) - manufacturing drawings or source code, etc.
- "as-built" baseline (aka Product baseline) - ready to test executables.
- "as-tested" baseline (post verification/validation).

Increasingly, models are built at all stages to communicate the user requirements, define system requirements, design and document system implementation, and to verify and validate the system. Establishing traceability between these different models can be challenging.

**The Role of the Systems Engineer.** The systems engineering process is usually comprised of the following seven tasks: State the problem, Investigate alternatives, Model the system, Integrate, Launch the system, Assess performance, and Re-evaluate. These tasks can be summarized with the acronym SIMILAR: State, Investigate, Model, Integrate, Launch, Assess and Re-evaluate. This Systems Engineering Process is shown in Figure 1. It is important to note that the Systems Engineering Process is not sequential. The functions are performed in a parallel and iterative manner. INCOSE, (2007).

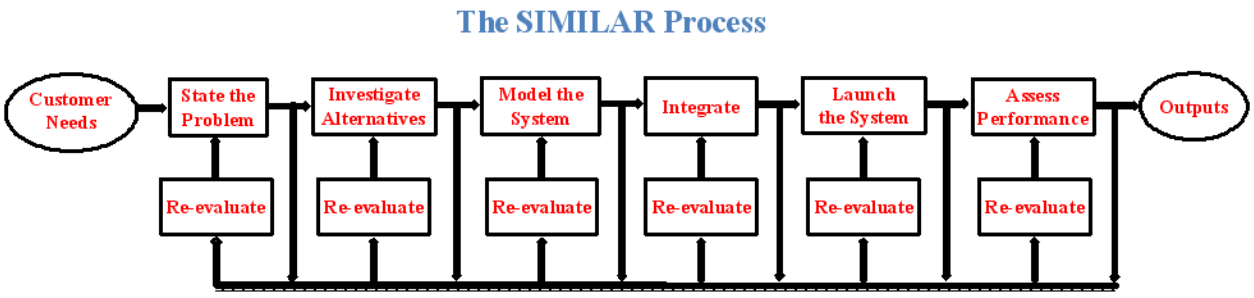


Figure 1. The Systems Engineering Process (Bahill, Gissing, 1998).

Part of this process is the allocation of the problem areas to engineering domains, (e.g. hardware engineering, mechanical engineering or software engineering etc). Each domain provides a method of solving a specific part of the overall system's requirements, and they are generally used in combination. Models are central to the exercise of trade-off analysis and evaluation of alternatives, as well as enabling communication throughout the process. In the rest of this paper, we will concentrate on the software domain as defined by UML.

## SysML and UML

In March 2003, the OMG issued a Request for Proposal (RfP) for a customized version of UML suitable for Systems Engineering written by the OMG Systems Engineering Domain Special Interest Group (SE DSIG). Friedenthal, Burkhart, (2003) gives early history on the development of the UML for SE RFP. The customization of UML for systems engineering is intended to support modeling of a broad range of systems which may include hardware, software, data, personnel, procedures and facilities. The goal is to provide a "standard modeling language for systems engineering to analyze, specify, design and verify complex systems, intended to enhance systems quality, improve the ability to exchange systems engineering information amongst tools and help bridge the semantic gap between systems, software and other

engineering disciplines" (OMG SysML, 2003). There was only one technology submission to the RfP, called OMG SysML. This has been accepted by the OMG and the finalization task force was completed in March, 2007 and version 1.0 officially issued in September 2007. The SysML Revision Task Force (RTF) was chartered at the OMG San Diego Meeting on March 30, 2007. The RTF will continue to propose refinements to the v1.0 specification for approval by the OMG. The plan will be to issue a minor revision (e.g., OMG SysML v1.1) in 2008.

For the sake of brevity, and the fact that the UML specification is 1000 pages and the SysML specification is 300 pages, we will not attempt to describe the two languages in detail. For more information on SysML, see OMG, (2007b) and Hause, (2006a). For more information on UML, see OMG, (2007a).

**Why model?** Before creating a model, several questions must always be asked. What question are we trying to answer? What purpose does the model serve? What is the extent of the model? How do we know when we are done? To prevent the length of this paper extending to that of a small book, we will limit ourselves to two examples: using SysML to define software requirements and ensure traceability, and using SysML to define the environment in which the software will be deployed. For both examples, the main requirement is that there will be clear and obvious mechanisms for establishing continuity and traceability between the systems and software models. At this point, it will be useful to include additional details on SysML mechanisms for creating relationships within and between model elements.

## *SysML Cross-Cutting Mechanisms*

Separation of concerns is the process of breaking a system into distinct features that overlap in functionality as little as possible (Dijkstra, 1974). A concern is any focus of interest in a system. Ideally, most components in a system will perform a single, specific function. Additionally, they often share common, secondary requirements with other system elements. These secondary requirements are said to cross-cut into the primary requirements. These are known as cross-cutting concerns. Examples include safety, traceability, timeliness, and risk. SysML supports two generic mechanisms for mapping/cross-cutting; these are requirements and allocation. Both define relationships that cross-cut standard separation of concerns. However, they differ in that allocation is a forward mapping mechanism (e.g. from system analysis to system design), whilst requirements is a backwards mapping mechanism (e.g. from the design model to the design requirements) in relation to the development lifecycle. For more information on cross-cutting concerns, see Hause, (2006b).

**Requirements.** A requirement specifies a capability or condition that must (or should) be satisfied. A requirement may specify a function that a system must perform or a performance condition a system must achieve. They are cross-cutting in that the requirements viewpoint can relate to all other viewpoints. Both SysML and UML provide a graphical means of embodying requirements, inherent in the model. SysML provides modeling constructs to represent text based requirements and relate them to other modeling elements. The requirements diagram can depict the requirements in graphical, tabular, or tree structure format. A requirement can also appear on other diagrams to show its relationship to other modeling elements. The requirements modeling constructs are intended to provide a bridge between traditional requirements management tools and the other SysML models. Relationships include derive, refine, satisfy, trace, verify, and copy. Figure 2 shows a requirements diagram for a Cruise Control system.

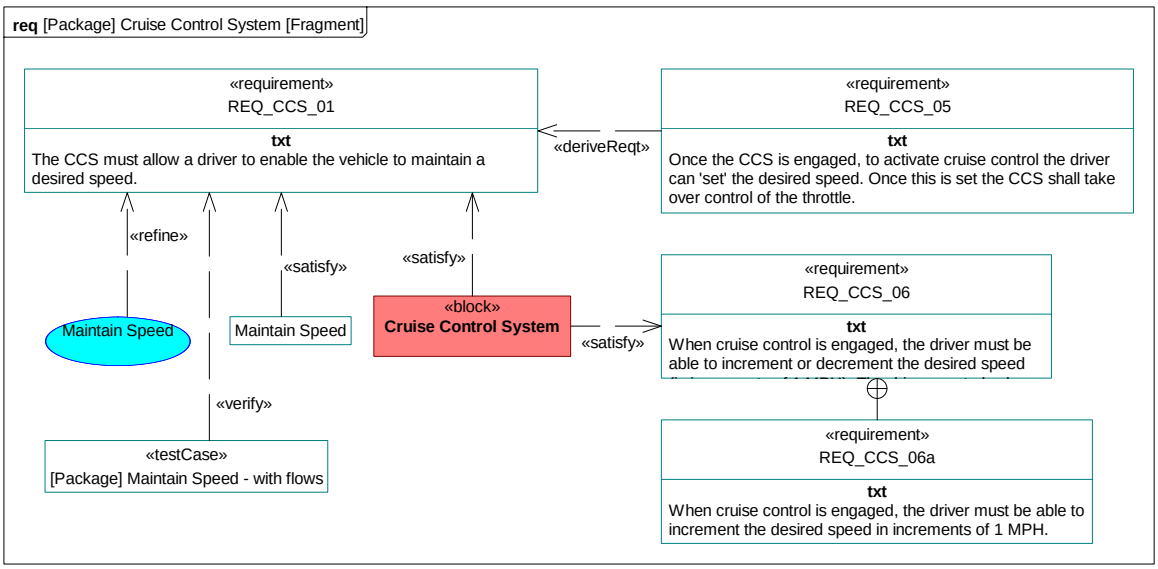


Figure 2. Requirements Example for a Cruise Control System

Requirement REQ_CCS_01 is satisfied by the Maintain Speed activity and Cruise Control System block, refined by the Maintain Speed use case, and verified by the Maintain Speed test case. REQ_CCS_005 is a derived requirement, and REQ_CCS_6a is a sub-requirement of REQ_CCS_6.

**Allocation.** Allocation is the term used by systems engineers to denote the organized cross-association (mapping) of elements within the various structures or hierarchies of a user model. The concept of "allocation" requires flexibility suitable for abstract system specification, rather than a particular constrained method of system or software design. System modelers often associate various elements in a user model in abstract, preliminary, and sometimes tentative ways. Allocations can be used early in the design as a precursor to more detailed rigorous specifications and implementations. The allocation relationship can provide an effective means for navigating the model by establishing cross relationships, and ensuring the various parts of the model are properly integrated. These take the form of internal links created in the model. SysML provides of means of displaying these on diagrams using call-out notation – notes that can be made visible on diagrams detailing the allocations. Cross-reference tables and matrices can also be generated based on the model information. Figure 3 shows an allocation table for a Cruise Control System.

## [Package] CC System [1]

| Allocated From | Relation | Allocated To |
|---|---|---|
| «part» CC Disp | Allocate | «Activity» Do Initialisation tests (CC Model::Analysis::Behaviour::Activities) |
| «Class» eCruiseControlPanel (CC Model::Design::Software::External Interface) | Allocate | «block» CC IO Card (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» eEMUIF (CC Model::Design::Software::External Interface) | Allocate | «block» CC IO Card (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» eTransmissionMonitor (CC Model::Design::Software::External Interface) | Allocate | «block» CC IO Card (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» eBrakePedalMonitor (CC Model::Design::Software::External Interface) | Allocate | «block» CC IO Card (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» cThrottle Controller (CC Model::Design::Software::Control) | Allocate | «block» CC Motherboard (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» pAccelerationProfile (CC | Allocate | «block» CC Motherboard (CC |

| | | |
|---|---|---|
| Model::Design::Software::Persistence Support) | | Model::Analysis::Structure::Vehicle::CC System) |
| «Class» pCalibration Manager (CC Model::Design::Software::Persistence Support) | Allocate | «block» CC Motherboard (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Class» cSpeedMonitor (CC Model::Design::Software::Control) | Allocate | «block» CC Motherboard (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Activity» Decrement Speed (CC Model::Analysis::Behaviour::Activities) | Allocate | «block» Cruise Control System (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Activity» Disengage CC (CC Model::Analysis::Behaviour::Activities) | Allocate | «block» Cruise Control System (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Activity» Do Initialisation tests (CC Model::Analysis::Behaviour::Activities) | Allocate | «block» Cruise Control System (CC Model::Analysis::Structure::Vehicle::CC System) |
| «Activity» Engage CC (CC Model::Analysis::Behaviour::Activities) | Allocate | «block» Cruise Control System (CC Model::Analysis::Structure::Vehicle::CC System) |

Figure 3. Allocation Table

The allocation table shows different types of allocation such as structural and functional. The activities are allocated to the structural elements, (Decrement Speed to the Cruise Control System), software to hardware, (pCalibration Manager to Motherboard), etc.

## *System to Software Handover*

**Requirements In, Requirements Out.** Here we are defining a *formal handover* as an agreement between system and software engineering. Even today, many organizations' development processes reflect their organizational structure. Some may say processes are often compromised by organizational structure to the extent that system and software engineers communicate (aka *argue*) through requirements specifications with protracted change management processes to further delay the successful handover. SysML does allow such formal handovers to take place but with one important difference. Often a software specification will contain too much detail and be lacking in rationale as to why decisions have been made. Modeling the software specification (i.e. a collection of SysML Requirements tracing back to the system design) within the system model and handing this model over to software engineering to augment with the software analysis and design, provides software engineering with the contextual information they were previously lacking. Used in the way SysML provides a model-centric approach to formal requirements handover. If the software engineering discipline is external from the system engineering organization, then the same mechanism of creating SysML requirements in the model can be performed and exported from the system model (in document form) and handed over to the software engineering organization. A good working relationship should be established between the two organizations to ensure issues can be resolved quickly.

**Requirements Traceability.** As stated previously, a software requirements specification of some description will normally be created as part of the systems engineering process. Integrating these requirements into the SysML/UML model will greatly improve traceability. <<Trace>> relationships can be added from the requirements to the source SysML model items indicating their origin. <<Satisfy>> and <<Refine>>relationships can then be added from the UML model items to the requirements to form an indirect link between the SysML and UML models, thus forming a less tightly coupled set of relationships.

**Object Oriented Modeling.** Using an object oriented approach; the system is decomposed into objects. Using encapsulation, each object is allocated responsibilities according to its main goal. Responsibility corresponds to the information it holds, its behavior, and the relationships it has with other objects to satisfy its goals. In UML, objects are grouped into classes containing attributes, operations, and associations and are modeled on the class diagram. In SysML, blocks

contain values, operations and associations and are modeled on the Block Definition Diagram (BDD). The internals of each block are then documented on an Internal Block Diagram (IBD). This includes its parts, ports, connectors, and flows. Another approach is to model system behavior independently. Activities can be identified and documented as a hierarchy or a flat structure. Activity diagrams can then be created to model the order of activities, conditional elements and the interchanges between the activities. In parallel or iteratively, the structure of the system is documented using the block diagrams described above. The activities can then be allocated to the blocks, and operations created for the blocks corresponding to those operations. Relationships between blocks can be added based on the requirement that the blocks communicate. Regardless of how it is done, the result is a set of blocks with behavior, attributes and relationships.

**Modeling software in SysML.** Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment, as well as software entities. Consequently, a block can represent a well defined area of functionality or structure in the system. In the following sections, we will examine the different ways of mapping from the behavior, structure and data defined in a SysML model to a UML model. Of course, any mapping will not be one to one, because the level of abstraction must necessarily be at a higher level. Functional, data, and structural software elements in the SysML model will not be the same as in a UML model. The job of the systems engineer is to define the requirements of the software. The job of the software engineer is to define a well-architected software solution. Consequently, refactoring will need to take place. However, as SysML is based on UML, we will at least be mapping between similar paradigms. An example of mapping from SysML block to UML software using allocation is shown in Figure 4.

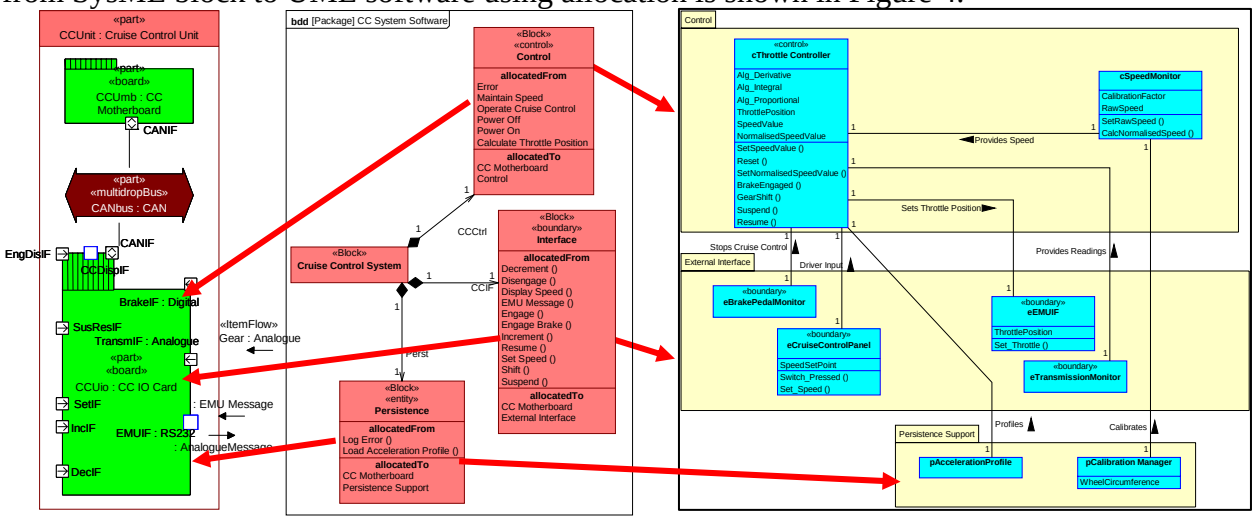


Figure 4. Mapping from Virtual Blocks to UML Packages and Hardware Platform.

In Figure 4, the Cruise Control system is shown as being made up of 3 blocks – Control, Interface, and Persistence. Allocated functionality is shown in the allocatedFrom compartment on the block and lists the activities allocated to the block. The allocatedTo compartment lists the deployment onto the hardware platform (CC Motherboard) shown to the left, and to the software packages which are shown on the right. The software packages are then further elaborated into classes to realize the required behavior. In this case, the blocks represent the strict separation of concerns common to robustness diagrams, which are Control, Boundary, and Entity. In this

simple example the mapping is straight-forward. This will not normally be the case. This example shows the system environment in terms of the hardware platform, and the software requirements as blocks.

## *Functional.*

The main behavioral aspects of UML and SysML are use cases, activities, operations, flows, events and signals, states, and (for coordinating these) system interaction diagrams.

**Use cases.** Use cases define testable system functionality from an outside-in perspective. Use cases can be mapped at the user, system and subsystem level (Cockburn, 2001). For large systems, use cases from the SysML model will not map directly to elements in the UML model. This is because the two models may be at different levels of abstraction. The SysML model may be concerned with the whole system of systems and the software model may only be concerned with the software part of a single system. Consequently, it is important to establish the use case scope. For example, a car may have the use case “Drive Vehicle”. This will involve the complex interaction of several systems described in sequence and activity diagrams, one of which may involve the operation of the cruise control system. The software engineer tasked with designing the cruise control software will then analyze the cruise control functional requirements and create a set of use cases scoped to the level of the cruise control software. For smaller systems, user and subsystem use cases describing the software functionality of a system may not need to be redefined by the software engineer.

**Activities.** Activities represent the basic unit of behaviour that is used in activity, sequence and state machine diagrams. Activity modeling emphasizes the inputs, outputs, sequences, and conditions for coordinating other behaviors. There are too many types and sub-types of activities and actions in the UML and SysML specifications to cover in a single paper, so we will consider the activity as a generic concept. The activity diagram is used to describe the flow of control and flow of inputs and outputs among actions. Activities will describe the functional characteristics of the structural elements (blocks, parts) to which they are allocated. In the UML model, they will map to use case capabilities of the systems corresponding to the SysML blocks. At the lower level, they could also map to classes and operations. In modelling the activities for a vehicle, the activity “Maintain Speed” is defined. The cruise control system will require several classes and operations in the software model to realise this behaviour. It is therefore useful to define a “Maintain Speed” use case for the software model to define the require software functionality. The “Maintain Speed” use case in the software model refines the “Maintain Speed” activity in the system model. On the other hand, the activity “update throttle” may only require an operation to send the throttle command from the cruise control to the engine management system and would not require a use case to describe its functionality.

**Block Operations.** Operations owned by a block correspond to services, capabilities, and/or functionality provided by the block. This will normally be implemented by a combination of the different domains that will map to the block. It will then be necessary to separate out the portions of the behavior that will be implemented in software. For blocks describing software intensive capabilities, again, this will depend on the level of abstraction at which the block is modeled, as described above in the activity section. As an operation normally involves a client server relationship, the requesting element will need to be considered. Often a single high level operation will map to the behavior described in an entire sequence diagram to realize the high level goal.

**Flows, Signals, and Events.** In SysML, flows can be typed by value types, data types, blocks and signals. These will correspond to the flow ports specified at each end of the connector describing the interface. These will be covered later in the section on hardware/software interfaces. Events are an indication that something has occurred in the system such as a message being sent or received that may potentially trigger effects by an object (OMG, 2007a). As these are atomic in nature, they will normally map directly to an event in the UML model. As they are normally connected to an effect, as described earlier, it will be necessary to consider the effect in the UML model. Signals are specified by a signal object; whose type represents the kind of message transmitted between objects, and can be dynamically created. The receipt of signals may be bound to activities, state machine transitions, or other behaviors. For communications systems, signals travel along an instance of a connector, originating in a required port and are delivered to a provided port. They can also be broadcast to multiple ports. Like events, signals can also be considered atomic and will usually map directly to the UML model. Likewise, it will also be necessary to consider any linked state transitions or behavior invocations.

**State Modeling.** State modeling will be problematic in that a SysML state machine may represent the behavior of a complete software system. It is in fact best practice to create this model for the software to be implemented as it can form the basis of the system lifecycle tests. However, the state machine as a whole will not map to a single class. The exact method for the mapping will correspond to the element that owns the state machine. For state machines modeled at a lower level of abstraction, they will map to 'controller' classes in the software model, as shown in Figure 5.

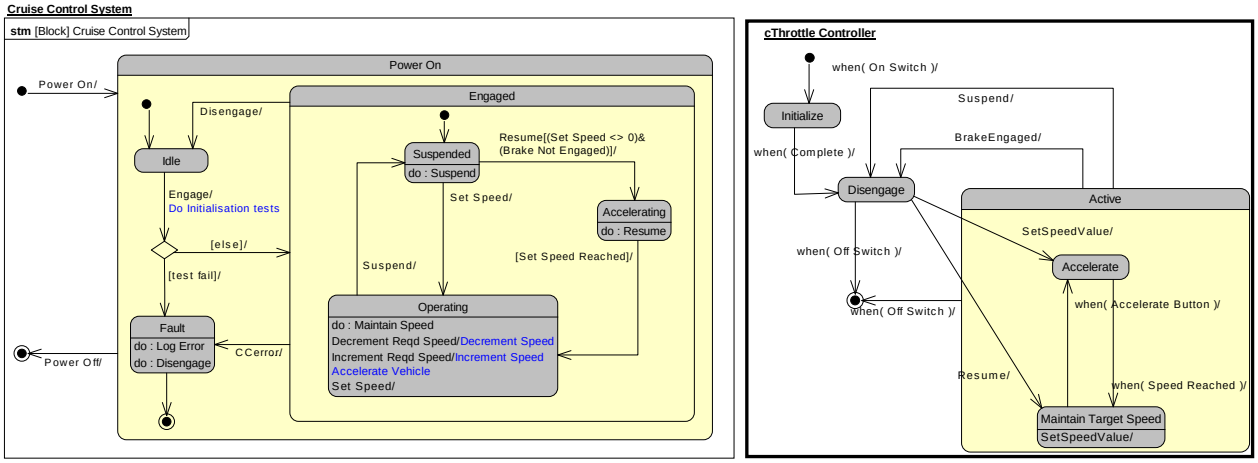


Figure 5. State diagrams for the Cruise Control System and Throttle Controller.

In this example, the state machine for the cruise control needs to take into account all the various states and sub-states of the system as a whole such as Power On and Disengaged. The Throttle Controller need only be concerned whether it is engaged or disengaged, and has the additional state of performing an initialization.

**System interactions.** System interaction diagrams in SysML are limited to Activity and Sequence diagrams. Communication diagrams are not currently included, and are not in fact currently possible as SysML has no instance model. Like the state machines, interaction diagrams will form the basis for system tests sequences, and will help to inform the UML model of required external behavior. Mapping to specific internal behavior of the required software is not normally applicable.

## *Structural.*

**Hardware/Software Interfaces and drivers.** SysML ports have done much to help in the definition of interfaces for both systems and software engineers. It is worth summarizing the usage of the different types of ports available in SysML. Required and provided interfaces (lollipops, cup and ball) associated with standard ports in a SysML model describe services that will be provided and required by the block owning the port. These are normally used in command and control systems, and when systems have a client-server relationship. This clearly identifies the relationship between the parts/blocks as active and resulting in control, rather than simple data transfer. The distiller model example developed during SysML has examples of this. Figure 6 shows an IBD of the distiller system and controller.

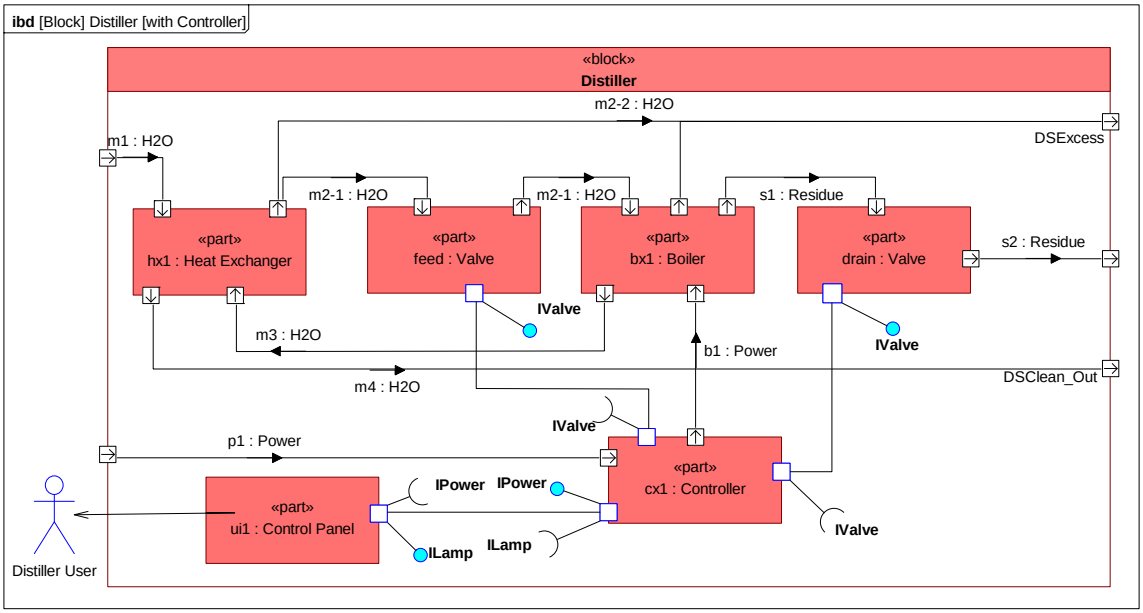


Figure 6. Distiller System With Controller.

Interfaces of this type also include Service Oriented Architectures (SOA), an increasingly prevalent architecture these days. Work is currently being done to integrate this concept into the UML Profile for DoDAF and MODAF. In general, standard ports and data type flow properties within flow specifications will map to ports on software parts. Note that classes used on SysML 'Standard' ports should be flowed-down as they are but may be further refined by s/w 'wrapping' a protocol around them for propagation throughout and out of the system.

For data intensive systems, data structures can be created on a Block Definition Diagram (BDD) using data types to both create message hierarchies and to then type the flow port on the Internal Block Diagram (IBD). Usually, this takes the form of an inheritance hierarchy with the highest level data type in the hierarchy typing the port. Modeling in this way means you can easily add new messages and you do not have to change the flow specification when a new message is added, as sub-types within an inheritance hierarchy can flow through a port typed by the parent type. This is particularly useful when modeling a system at the level of needlines and where the item flows model the information exchanges. These ports can later be allocated to the communications ports typed as Ethernet, RS-232, etc. described further on.

Looking at value versus block for typing ports, it will depend on the context and purpose of the definition. Consider the example of the distiller system where the flows specify the water in

various forms is flowing through the system. To evaluate the high level system to spell out how the system works, fluid can be defined as a block in its various forms to type the flows and the ports between the different parts. This gives a clear view of how the system as a whole is meant to work. This is useful for mechanical engineers. One also needs to be aware for example, of whether the different parts will be reused. Take the valve for example; typing the ports on the valve by residue or H2O means that only one type can flow through them. Creating an inheritance hierarchy of fluid, sub-typed by H2O and residue, and typing the ports by fluid means the either H2O or residue can flow through them shown in Figure 6.

If we then turn to how to monitor and control the system, we need to look at additional characteristics of the system, for example temperature, flow rate and so on. The systems engineer can then type this information as SI units to communicate with both the software and hardware engineers, also specifying the level of precision that is needed for the telemetry used to measure the system. The telemetry can then be specified as analog, digital, pulse counters and so on, and the appropriate telemetry can be acquired that meets the job. The software engineer can then use these definitions to define software conversion algorithms and types to process this telemetry. Figure 7 shows the Cruise Control external interfaces with analogue, digital, RS232, and other interfaces.

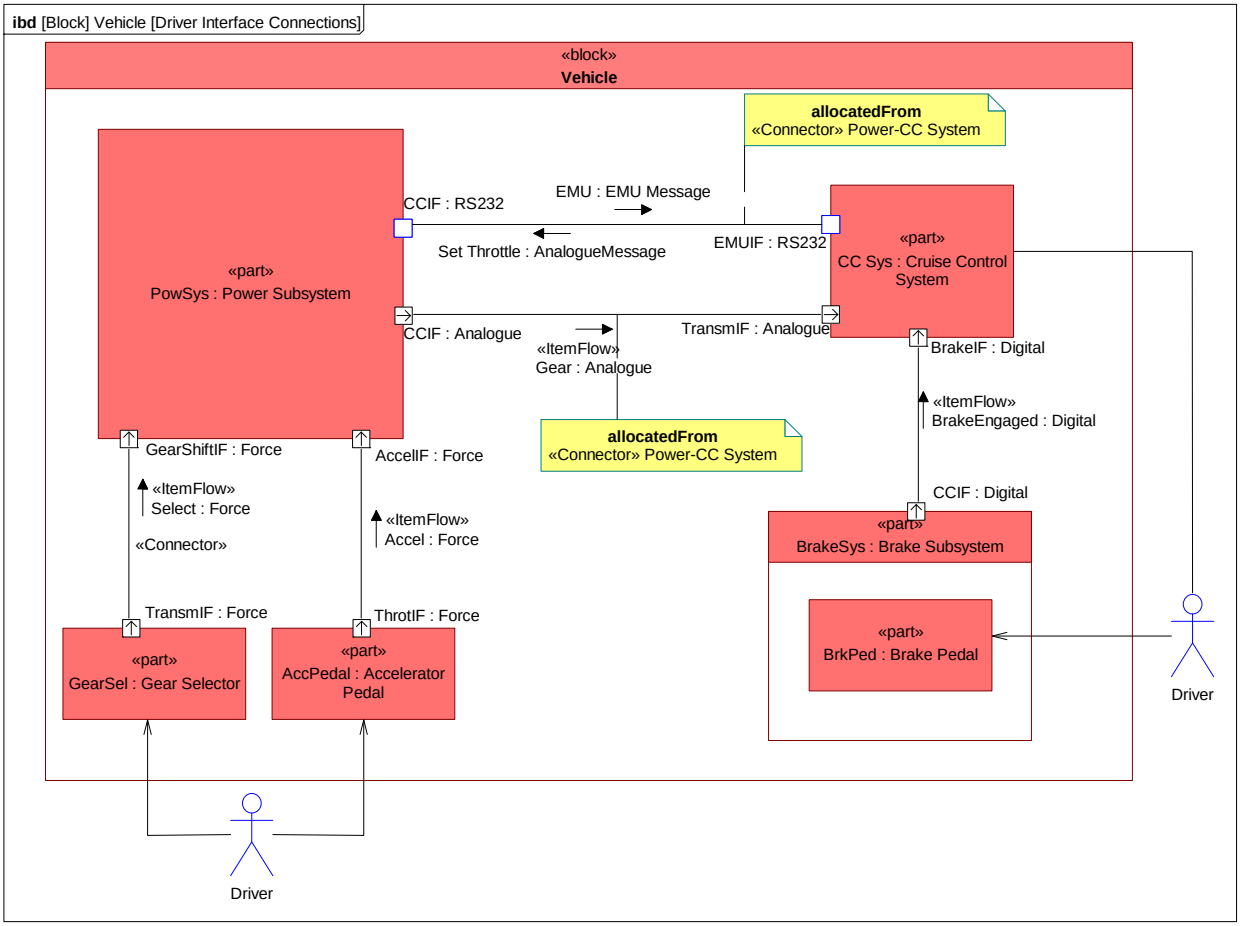


Figure 7.Cruise Control External Interfaces.

**User Interface.** Hause, Thom (2007a), described in detail how Human Computer Interface (HCI) characteristics can be integrated into SysML and architectural framework models. As this

subject is more than sufficient for a paper, if not a book, it would be best refer to that paper and its references for more information on the subject. Other papers on this subject were also presented at INCOSE IS 2007 including Bruseberg, (2007) and McKenna, (2007).

## *Other Aspects*

**Data.** The specification of data sets is a contentious issue between systems and software engineers. This often takes the form of tables at the back of the specification that are difficult to translate into classes, attributes, and associations in a UML model. An ideal method for modeling data is the use of SysML data types to create a diagrammatic view of the data that is then used as the basis for automatic generation of the aforementioned tables. Of course, this does not mean to imply that software engineers are to dim to read a table and translate it into a class diagram. Instead, it suggests that the exercise is not necessary.

## *Resolving System and Software Architectures*

**Architecture-centric.** Up until now, we have concentrated on the flow down of information from Systems Engineering into Software Engineering. Increasingly there are situations where either legacy or mandated software architectures exist that must be respected by systems engineers. Software architectures exist for different reasons than a system architecture e.g. performance-efficiency, maintainability and reusability. Typically these constraints are imposed on software engineering (sometimes self imposed) which are orthogonal to the constraints (e.g. requirements) placed on software engineering by systems engineers. Whilst systems engineers may also have to respect some of these constraints if they are required by the system stakeholders, they are not always required at the system level. System-level reuse is relatively immature compared to software reuse and is rarely asked for by a stakeholder. There are situations where what is perceived to be software architecture is in fact system architecture. An example is the evolving architectures used in Integrated Modular Avionics (IMA). IMA imposes a high degree of standardization, modularity and reuse of hardware (primarily to address hardware obsolescence) whilst offering enormous flexibility in the deployment of software. However, this flexibility should not only be exploited by software engineers, it is something that systems engineers must also exploit. The logical (behavioral) aspects within the system model should encompass and embrace the architecture called-up in the IMA standards. Figure 8 shows an example of IMA architectural elements.

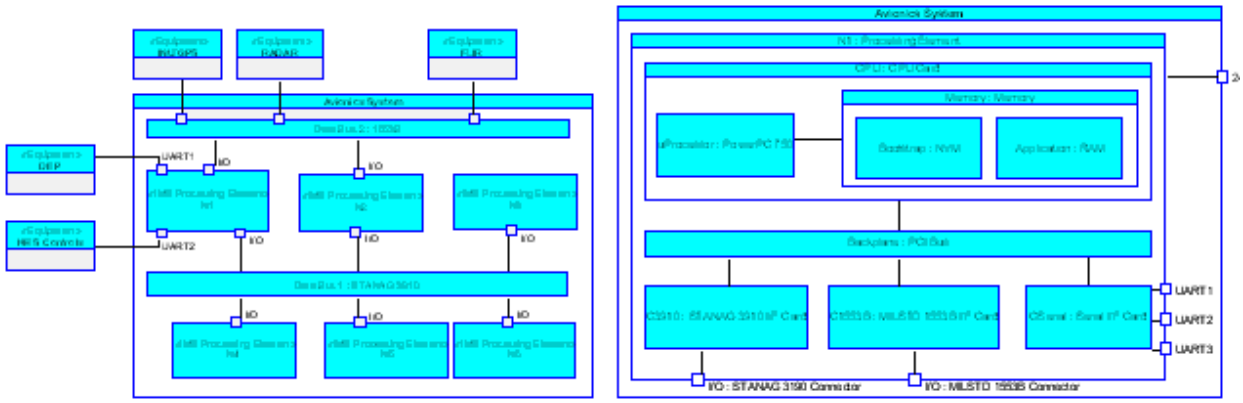


Figure 8. IMA Topology of Processing Elements and Internals of a Processing Element.

Systems engineers that ignore software-intensive architectures run the peril of seriously complicating (if not compromising) traceability of information from the system model into the software model. SysML, being closely aligned and underpinned by the more software oriented UML, is an ideal vehicle for system and software engineers to co-develop an implementation of an overarching software-intensive architecture. Similar examples of this exist in the automotive industry under AUTOSAR and their proved success may result in analogous architecture-centric approaches being developed in other domains. See Thom, Hause (2006) for more information on IMA, and Korff (2006) for more information on AUTOSAR.

**Software Architecture in the System Model.** System architectures such as IMA and AUTOSAR will result in the system behavioral model, if correctly documented, being structured into software-intensive SysML activities (e.g. representing partitions). This allows systems engineers to analyze these from many perspectives including performance-efficiency, safety, and to assess deployment configurations, activities that should not be left for software engineering to resolve. We will now focus on the mapping of information from a system model into an established i.e. legacy software architecture. It must be noted that the term *legacy* is used in its broadest sense meaning both an existing software implementation being updated by new requirements and a non-implemented architecture that is selected or desired for implementation – i.e. the choice of the software architecture has been made and it is now a *legacy* decision. In the situation of legacy architectures, system and software engineers must collaborate in developing an abstraction of the software architecture within the system model. This is best represented as UML artifacts (e.g. UML classes) structured to reflect the software architecture and then using all the allocation and traceability links provided by SysML (described above).

## *The Importance of Process.*

The SysML allocation, traceability and other cross cutting mechanisms presented in this paper are not meant to replace a well written requirements specification. Rather, they are meant to enhance the traceability and impact analysis techniques already available to systems engineers. They provide a means of clarifying the intent of the requirements, as software engineers will already be familiar with most of the concepts behind the SysML models. As system and software development is iterative in nature, problems found in the software development phases can more quickly be fed back to the systems engineers for resolution. A good starting point for defining a process or integrating these concepts into an existing process is the Object Oriented Systems Engineering Methodology (OOSEM). This has been successfully adopted by several major companies. For more information, see Lykins, et al, (2000) and other information available at the OOSEM website *http://syseng.omg.org*.

**Helpful Hints.** However, the mapping is done, it is imperative that a well defined process be specified elaborating how the mapping fits in to the overall process, whether it is suggested or mandatory, and how updates, modifications, change notes, etc will be handled. Indicators as to the level of abstraction and the nature of the SysML elements will also be helpful to the software engineer. For example, a set of stereotypes associated with blocks ports and connectors indicating whether they are hardware, software, virtual or other can help in the allocation of domains and clarify the intent.

**Adoption.** Compared with UML, SysML is a relatively new language and although starting to be widely used, it will not have been used for an entire development lifecycle for any major projects. SysML has the advantage of its close association with INCOSE and the fact that most

other notations available are either proprietary, or impractical for use on large scale projects. As with the integration of all new concepts, methodologies, and techniques, it is important to prototype the use of SysML on a non-critical project prior to full adoption. This ensures that lessons can be learned on how best to use SysML and how it can help communicate with both the stakeholders of the project as well as software, hardware, mechanical, etc engineers further downstream in the development lifecycle.

So, will software engineers look at all this as systems engineers meddling in their area and telling them how to design software? Given the level of internal competition and conflict, I have found on some projects, it is a question worth asking. A well defined process can help. It should instruct the systems engineers how to describe the software functional requirements and constraints, and (when specified by the requirements) the architecture without straying into software design. Most importantly, it will tell them when to stop. Speaking as a systems engineer with many years of software engineering experience, I wish I had these capabilities thirty years ago when I started building systems. It is certainly preferable to the "Death By Words" approach to requirements or having to re-factor a functional decomposition model into an object oriented or component based approach.

## Conclusion

This paper has spelled out how SysML to UML can be used together. Used effectively, they can increase communication between members of development teams, enable traceability, and manage change by providing mechanisms to analyze the impact of changes. They are based on project experience, discussion in training courses on SysML presented to students, and discussions at conference. SysML is a relatively new tool and techniques and uses for it are being investigated on an ongoing basis. It is hoped that the adoption of these techniques will improve system development, reduce miscommunication, reduce rework and clarify the true impact of system changes. As always, the authors would welcome feedback from any project that has implemented these techniques.

## References


Bahill, A.T., B. Gissing, B., 1998, Re-evaluating systems engineering concepts using systems thinking, IEEE Transaction on Systems, Man and Cybernetics, Part C: Applications and Reviews, 28 (4), 516-527, 1998.

Box, George E. P.; Norman R. Draper (1987). Empirical Model-Building and Response Surfaces, p. 424, Wiley. ISBN 0471810339, Page 424.

Bruseberg, A. 2007. Human Factors Integration for MODAF: Needs and Solution Approaches, June, 2007 INCOSE International Symposium 2007 Proceedings.

Cockburn, A., 2001, Writing Effective Use Cases, Addison, Wesley, Pearson Education, ISBN 0-201-70225-8

Dijkstra, Edsger, 1974, On the Role Of Scientific Thought, accessed online November, 2005 from http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html

DoD Architecture Framework Version 1.0 Volume I: Definitions and Guidelines 30 August 2003

Dsouza, D. (2001). Model-Driven Architecture and Integration, Opportunities and Challenges Version 1.1, Available from www.omg.org. [Accessed November, 2006].

Friedenthal, S., Burkhart, R. “Extending UML From Software To Systems,” Proceedings Of The INCOSE 2003 International Symposium, 2003.

Hause, M.C., 2006a, The Systems Modeling Language - SysML, Sept 2006, INCOSE EuSEC Symposium 2006 Proceedings.

Hause, M. C., 2006b, Cross-Cutting Concerns and Ergonomic Profiling in UML/OMG SysML, July 2006, INCOSE International Symposium 2006 Proceedings.

Hause, M.C. and Thom, F., 2007a, HCI Aspects of SysML and Architectural Frameworks, June, 2007 INCOSE International Symposium 2007 Proceedings.

Hause, M.C. and Thom, F., 2007b, Bridging the Chasm – Tracing from Acrchitectural frameworks to SysML, June, 2007 INCOSE International Symposium 2007 Proceedings.

IEEE, (2000). "Recommended Practice for Architectural Description for Software-Intensive Systems". Appears in IEEEstd 1471-2000. Available from www.ieee.org [Accessed November, 2006].

Illingworth, V., Glaser, E.L., Pyle, I.C., 1991, Oxford Reference Dictionary of Computing, Oxford University press, Walton Street, Oxford, 1991.

INCOSE, 2007, http://www.incose.org/practice/whatissystemseng.aspx Available online, Accessed November 2007

Lykins, H., Friedenthal, S., And Meilich, A. Adapting UML For An Object-Oriented Systems Engineering Method (OOSEM)”, Tenth Annual Int Symp INCOSE proceedings. Minneapolis, Mn, USA, July 16-20, 2000.

Korff, A., 2006, AUTOSAR and SysML – a natural fit, 3rd European Congress ERTS – EMBEDDED REAL TIME SOFTWARE, 25, 26 & 27 January 2006 – Toulouse, France

McKenna, B., Expanding Functional Analysis to Develop Requirements for the Design of the Human-Computer Interface, June, 2007 INCOSE International Symposium 2007 Proceedings.

OMG SysML™ Object Management Group (OMG), 2003, UML™ for Systems Engineering Request for Proposal OMG Document: ad/03-03-41, Available from www.omg.org. [Accessed September 2003].

Object Management Group (OMG), 2007a. Unified Modeling Language: Superstructure version 2.1.1 with change bars ptc/2007-02-03. [online] Available from: http://www.omg.org [Accessed September 2007].

OMG Systems Modeling Language (OMG SysML™), V1.0, 2007b, OMG Document Number: formal/2007-09-01, URL: http://www.omg.org/spec/SysML/1.0/PDF, Accessed November, 2007

Sommerville, I. And Sawyer, P., Requirements Engineering, A good practice guide, John Wiley and Sons, June 2000.

Thom, F., Hause, M.C., 2006, Modeling Distributed Integrated Modular Systems Using the UML™ and the SysML™. 3rd European Congress ERTS – EMBEDDED REAL TIME SOFTWARE, 25, 26 & 27 January 2006 – Toulouse, France

# Biographies

**Matthew Hause, Chief Consultant - Artisan Software Tools**

Matthew has been developing real-time systems for almost 30 years. He started out working in the Power Systems Industry, and has been involved in Process Control, Communications, SCADA, Distributed Control, Defence and many other areas of real-time systems. His roles have varied from project manager to developer. His role at Artisan includes mentoring, sales presentations and training courses. He has written a series of white papers on project management, Systems Engineering, and systems development with UML and SysML. He has been a regular presenter at INCOSE and several other international conferences.

**Francis Thom, Principal Consultant - Artisan Software Tools**

Francis has been developing real-time systems for the past 20 years. He started developing C3I Systems for Surface and Sub-Surface vessels. He has also worked on multi-role military and commercial avionics systems, on safety-critical railway signaling systems and On-Board Computers for space rockets. His roles have varied from team leader to developer. Francis has also worked on Process Improvement initiatives for a number of companies. His role at Artisan includes mentoring, sales presentations and training courses. He has written a series of white papers on Process Improvement, Systems and Software Engineering. He has been a regular presenter at INCOSE and several other international conferences.