32nd Annual INCOSE international symposium
hybrid event
Detroit, MI, USA
June 25 - 30, 2022

You Can't Touch This! - Logical Architectures in MBSE and the UAF

Matthew Hause
Systems Solutions Inc (SSI)
3208 Misty Oaks Way, Round Rock, Texas, USA
+1 917 514 7581
mhause@systemxi.com

Lars-Olof Kihlström
Syntell AB
PO Box 10022, SE-10055 Stockholm, Sweden
+46 706661978
lars-olof.kihlstrom@syntell.se

Abstract. Logical or abstract architectures are an essential concept in systems engineering. They are included in the systems engineering handbook, the OOSEM process, the SEBOK, several modeling languages, and the ISO 15288 process definition. A logical architecture is a solution-independent model of the problem domain used to understand "what" needs to be done, while avoiding defining "how" it will be done. The logical architecture includes all the related logical elements without constraining the architecture to a particular technology or environment. It traces to the physical architecture which defines how to implement the architecture using specific technologies. Logical architectures can be defined using MBSE languages such as the systems modeling language (SysML). They are implicit in the Operational set of views in architecture frameworks such as DoDAF, MODAF, NAF and their implementation in UAF using SysML. NAF has recently changed the title of the Operational views to Logical views to further emphasize the purpose of the views. This paper will define the benefits of using a logical architecture and provide guidance on how it can be implemented.

Introduction

The logical architecture is a model that is used to provide a detailed description of the system without defining the system technology or environment. The title of the paper refers to the fact that the logical architecture does not contain solution specific elements, i.e., it is comprised of non-physical things or things that cannot be touched. The rule of thumb often quoted is "if you can hit it with a hammer, it should not be in the logical model." While a good rule of thumb, there are exceptions in terms of what the Unified Architecture Framework (UAF) refers to as "known resources", which will be covered later in the paper. Essentially, selection of the contents of a logical architecture is a balancing act that needs careful consideration, especially when an object-oriented approach to the model is applied.

What is Meant by a Logical Architecture?

The INCOSE Systems Engineering Handbook (INCOSE, 2015) states "Logical models, also referred to as conceptual models represent logical relationships about the system such as whole-part

relationship, an interconnection relationship between parts, or a precedence relationship between activities to name a few." It further discusses the development of different architecture viewpoints and to "Select, adapt or develop models of the candidate architectures of the system such as logical and physical models. It is sometimes not necessary nor sufficient to use logical and physical models." For example, the UAF also provides Strategic/Capability Views and Services Views that both provide a solution independent expression of stakeholder requirements and solution independent services specifications. "The models to be used are those that best address key stakeholder concerns. Logical models may include the functional, behavioral, or temporal models." (INCOSE, 2015)

The Systems Engineering Body of Knowledge (SEBOK) states that "the logical architecture defines system boundary and functions, from which more detailed system requirements can be derived. The starting point for this process may be to identify functional requirements from the stakeholder requirements and to use this to start the architectural definition, or to begin with a high-level functional architecture view and use this as the basis for structuring system requirements. The exact approach taken will often depend on whether the system is an evolution of an already understood product or service, or a new and unprecedented solution. However, when the process is initiated, it is important that the stakeholder requirements, system requirements, and logical architecture are all complete, consistent with each other, and assessed together at the appropriate points in the systems life cycle model." (INCOSE, 2021) The different elements defined in this architecture help to approach the problem from many different perspectives and defining them, assembling these together, and adding traceability between them identifies inconsistencies and spurs innovation.

Chapter 16 of Friedenthal et al (2008) describes the definition of the logical architecture using the Object-Oriented Systems Engineering Methodology (OOSEM) as follows: "This activity is part of the system architecture design that includes decomposing the system into logical components that interact to satisfy system requirements. The logical components are abstractions of components that implement the system, which perform the system functionality without imposing implementation constraints. An example of a logical component is a user interface that may be realized by a web browser or display console or an entry/exit sensor that may be realized by an optical sensor. The logical architecture serves as an intermediate level of abstraction between the system requirements and the physical architecture that can reduce the impact of both requirements and technology changes on the physical design." (Friedenthal, 2008)

Logical Architecture Development

So how are these logical components defined and developed? This understandably follows normal systems engineering best practice in that it is driven by functional requirements or behavior. What functionality does the system need to provide to its stakeholders? "OOSEM provides guidelines for decomposing the system into its logical elements. Functions for logical elements are derived from logical scenarios to support black system functions. Logical element functionality and data may be repartitioned based on other criteria such as cohesion, coupling, design for change, reliability and performance." (INCOSE, 2015) So, having defined the required stakeholder needs, use cases, user stories or scenarios, the functions, activities, or behavior identified in these scenarios is used to define logical components.

Schindel (2021) in training presentations describing the S*Models and Patterns and the Agile Systems Engineering Lifecycle Management (ASELCM) defines logical architectures describing interactions. “A Functional Role (or simply Role) is the behavioral part played by a system component in an Interaction. It is the input-output behavior seen by the other roles in the interaction. Every Interaction must have two or more Roles—else it would not be an Interaction! A functional role is described entirely in behavioral terms. These are also called “logical systems”. One of the uses of containment hierarchy for roles / logical systems is that we can build up “larger” logical systems (behaviors) that contain (interconnected, interacting) “smaller” logical systems behaviors). At any given level of such a hierarchy (for example, the whole vehicle level), we can view the interacting smaller logical systems one level down, understanding the “logical architecture” described by the relationships summarizing their interactions.” (Schindel, 2021)

Allocation of Logical to Physical

The defined logical model can and must be allocated to the physical architecture to define a workable solution. “Defining a physical structural model of the architecture of a system consists of identifying system elements capable of performing the functions of logical models, identifying the physical interfaces capable to carry input/output flows and control flows, and taking into account architectural characteristics that characterize the system in which they are included.” (INCOSE, 2015) This makes the abstract concrete and defines “How” the system will work.

Benefits of Logical Architectures

Logical architectures provide the following benefits:

- To ensure that the system design meets the stakeholder requirements
- Providing a bridge from requirements to the solution
- Preventing “solutioneering” rather than engineering
- Capturing the main concepts of the architecture prior to defining a solution
- Reflect the true location of the system of interest (SOI) in relation to stakeholders and other systems
- Define an objective set of concepts and measures for trade-off analysis of solutions
- Providing business objects for service definition
- Bridging the gap between capabilities and implementation
- Facilitating impact analysis and traceability
- Minimizing the effect of changes in the architecture.
- Spurs innovation by forcing a rethink of the problem

Logical architectures also have a significant role to play when dealing with system of systems and an example of this will be given later in this paper. The paper will look at the importance of the logical architecture, traceability to the other views and the trade-offs involved in creating the logical architecture. They are not sufficient without the physical model of course. Otherwise, systems would never actually be built!

Logical Architecture in Modeling languages

The Systems Modeling Language (SysML) is the most widely used standardized systems modeling language and notation. It is used to model systems in both the abstract and concrete (logical and physical) views that include behavioral, structural, parametric and requirements views.

(OMG, 2017). SysML includes an allocation relationship to represent the allocation of functions to elements, allocation of logical to physical elements and other types of allocation. SysML has no predefined set of views to differentiate a logical vs. a physical architecture, levels of abstraction or even levels of detail. In practice this is done using different package structures, custom stereotypes, custom graphics, or a combination of all of these. The Harmony Process by IBM (Douglass, 2014) describes a means of defining a logical architecture as well as allocation to a physical architecture. Architecture frameworks employ a means to define standardized views that separate the logical from the physical, as will be illustrated in the next section describing UAF.

The Unified Architecture Framework (UAF)

For enterprise modeling, an architecture framework is required to understand systems of systems and how they change over time. DoDAF is the Department of Defense Architecture Framework (DoD, 2012) and MODAF is the Ministry of Defence Architecture Framework (MOD, 2020). NATO created NAF version 3 (NATO Architecture Framework) based on MODAF and has recently adopted NAF version 4 (NATO, 2018). NAF version 4 has been adopted by many European countries including the UK. The Unified Architecture Framework (UAF) is built on top of SysML and is used to define the overall goals, strategies, capabilities, interactions, standards, operational and systems architecture, systems patterns and so forth (UAF, 2019). Figure 1 shows the UAF Grid where rows are viewpoints, and columns are different forms of representations or different modeling aspects (state diagrams, activity diagrams, block diagrams, etc.).

UAF	Motivation Mv	Taxonomy Tx	Structure Sr	Connectivity Cn	Processes Pr	States St	Sequences Sq	Information If	Parameters Pm	Constraints Ct	Roadmap Rm	Traceability Tr
Architecture Management Am	Architecture Principles Am-Mv	Architecture Extensions Am-Tx	Architecture Views Am-Sr	Architectural References Am-Cn	Architecture Development Method Am-Pr	-	-	Dictionary Am-If	Architecture Parameters Am-Pm	Architecture Constraints Am-Ct	Architecture Roadmap Am-Rm	Architecture Traceability Am-Tr
Summary & Overview Sm-Ov												
Strategic St	Strategic Motivation St-Mv	Strategic Taxonomy St-Tx	Strategic Structure St-Sr	Strategic Connectivity St-Cn	Strategic Processes St-Pr	Strategic States St-St	-	Strategic Information St-If	**Environment En-Pm and Measurements Me-Pm and Risks Rk-Pm**	Strategic Constraints St-Ct	Strategic Roadmaps: Deployment, Phasing St-Rm-D, -P	Strategic Traceability St-Tr
Operational Op	**Requirements Rq-Mv**	Operational Taxonomy Op-Tx	Operational Structure Op-Sr	Operational Connectivity Op-Cn	Operational Processes Op-Pr	Operational States Op-St	Operational Sequences Op-Sq	Operational Information Model Op-If		Operational Constraints Op-Ct	-	Operational Traceability Op-Tr
Services Sv		Services Taxonomy Sv-Tx	Services Structure Sv-Sr	Services Connectivity Sv-Cn	Services Processes Sv-Pr	Services States Sv-St	Services Sequences Sv-Sq			Services Constraints Sv-Ct	Services Roadmap Sv-Rm	Services Traceability Sv-Tr
Personnel Ps		Personnel Taxonomy Ps-Tx	Personnel Structure Ps-Sr	Personnel Connectivity Ps-Cn	Personnel Processes Ps-Pr	Personnel States Ps-St	Personnel Sequences Ps-Sq	Resources Information Model Rs-If		Competence, Drivers, Performance Ps-Ct-C, -D, -P	Availability, Evolution, Forecast PS-Rm-A,-E,-F	Personnel Traceability Ps-Tr
Resources Rs		Resources Taxonomy Rs-Tx	Resources Structure Rs-Sr	Resources Connectivity Rs-Cn	Resources Processes Rs-Pr	Resources States Rs-St	Resources Sequences Rs-Sq			Resources Constraints Rs-Ct	Resources Roadmaps: Evolution, Forecast Rs-Rm-E, -F	Resources Traceability Rs-Tr
Security Sc	Security Controls Sc-Mv	Security Taxonomy Sc-Tx	Security Structure Sc-Sr	Security Connectivity Sc-Cn	Security Processes Sc-Pr	-	-			Security Constraints Sc-Ct	-	Security Traceability Sc-Tr
Projects Pj	-	Projects Taxonomy Pj-Tx	Projects Structure Pj-Sr	Projects Connectivity Pj-Cn	Projects Processes Pj-Pr	-	-	-		-	Projects Roadmap Pj-Rm	Projects Traceability Pj-Tr
Standards Sd	-	Standards Taxonomy Sd-Tx	Standards Structure Sd-Sr	-	-	-	-	-		-	Standards Roadmap Sd-Rm	Standards Traceability Sd-Tr
Actual Resources Ar	-	-	**Actual Resources Structure Ar-Sr**	**Actual Resources Connectivity Ar-Cn**	**Simulation**			-	-	**Parametric Execution/ Evaluation**	-	-

Figure 1. The Two-Dimensional Grid of View Specifications in UAF

Security and human factors (personnel) views were added to the UAF to improve the coverage of these areas of concern. DoDAF/MODAF Systems views were renamed the Resources views as they use systems, software, personnel, natural resources, etc. The UAF fully implements DoDAF,

MODAF, and NAF and is in fact a superset of these frameworks. The UAF was previously called the Unified Profile for DoDAF and MODAF (UPDM) and was ratified by the Object Management Group (OMG). Several papers have been written on the UAF and its support of SoS modeling including (Hause, Dandashi 2015) and (Hause 2014). The full details of SysML and UAF are not included here for space reasons. Please see the above references for more information. The UAF provides the following viewpoints and concepts:

- Strategic Capability and Enterprise Concepts: defines the "why" and "what" and "when" before the "how"
- Operational Logical Architecture: Defines the enterprise architecture in a solution independent form.
- Services Concepts: solution independent definition of enterprise services (producing and consuming) and traceability to capabilities, operations and implementing resources
- Human Factors: How people and systems interact, and their expected knowledge & skills
- Security: Identifying risk, its mitigation, and integrating security into the architecture
- Standards: definition of and compliance with standards in the architecture
- Project Deliveries: phased milestone approach to capability deployment
- System Configuration Over Time: deployment and changes in roadmaps and timelines
- Tie-in to Non-System Elements in the Architecture: Easy way to link the entire Architecture to Requirements
- Built-in Traceability Between Multiple Views: Between Layers and Across Layers (OMG, 2022)

The UAF Operational Views

The operational architecture is a "logical" architecture in the sense that physical implementation decisions are deferred to downstream architecture decisions and tradeoffs (a solution-independent architecture). Logical in this sense means connecting ideas in a sensible way, based on the rules of logic or formal argument. In other words, the logical architecture is what reasonably "follows" from the drivers, challenges, opportunities, desired effects, and capabilities defined in the UAF strategic or capability views. (Martin 2020) These also include associated measures, visions, goals, and their strategic context. These are used to define associated concepts of operation using logical (unimplemented) operational agents, that perform activities that compose an operational architecture element which provides those capabilities. The main elements in the operational views are the following:

- Operational Architecture – a type used to denote a model of the Architecture, described from the Operational perspective (Note: this represents a large composition or aggregation of operational performers, that is described from the operational perspective).

- Operational Performer – a logical entity that Is Capable to Perform Operational Activities which produce, consume and process Resources.

- Operational Activity – an activity that captures a logical process, specified independently of how the process is carried out (Note: an activity may contain a view of a logical process flow) (OMG, 2022)

An Operational Architecture defines operational behavior elements (e.g., processes, states, sequences) and allocates these to operational structure elements (e.g., operational architectures, performers, known resources to be used in the operational setting, roles, connectors, exchange items, exchanges, ports, and interfaces). The behavior elements define operational expectations for the operational activities that map to capabilities that will in turn help achieve enterprise goals.

All the elements in the operational architecture are solution-independent except for "known resources", which require some explanation. They are essentially to be considered as boundary conditions, i.e., systems that the system under consideration interacts with but where the system of interest needs to be adapted to accommodate it since each of the known resource's behavior is already defined. (See elements shown in Figure 10.) Consequently, it is essential that any known systems that must be part of the eventual architecture are identified early on to ensure that the system will meet the stakeholder requirements. Making the system too abstract or "too logical" and ignoring these systems can result in an artificially unconstrained system, that is not practical or will not interface to the required environment. Instead, by integrating these systems into the logical architecture, issues can be identified early and resolved. If the constraints imposed by a known system prevent the logical architecture from satisfying stakeholder needs and requirements, then changes to that known system will also be required.

Example Automotive Factory Model

Problem Statement: Powerhouse Engines (PE Inc.) is an automotive supply company providing internal combustion engines. PE Inc. finds that it has gradually become less competitive over the years largely due to their outdated technology and largely manual processes. Foreign and domestic competitors have started to cut into their business and the stakeholders are concerned that the company's loss of market share will accelerate and that they will eventually become insolvent. To combat this, the shareholders have proposed an investigation into strategies and technologies such as Augmented reality, Robotic assembly systems, 5G, AI, Additive manufacturing, outsourcing of select manufacturing and IT systems, Battery technology, Data analytics, Hybrid/electric engines, etc. These technologies will be rolled out over a 3-phase technology deployment plan.

Capturing the Main Concepts of the Architecture

The High-Level Manufacturing Concept diagram is shown in Figure 2. The purpose of the diagram is to describe the main enterprise concepts in a manner that is easy to understand to ensure a common understanding. The main elements identified in the problem statement are illustrated below. To a large extent they are defined in a manner that is solution independent. For example, the part supplier shown in the upper left-hand corner could be an outsourced external company, an internal casting department, or a 3D printer located in house. All of these are valid part suppliers, and each will have its advantages and disadvantages regarding supply chain delays, cost, flexibility, etc. In fact, all will be deployed over the 3 phases of technology introduction. Other elements shown in the context are the product itself, the workers, assembly line, parts, transportation, etc.

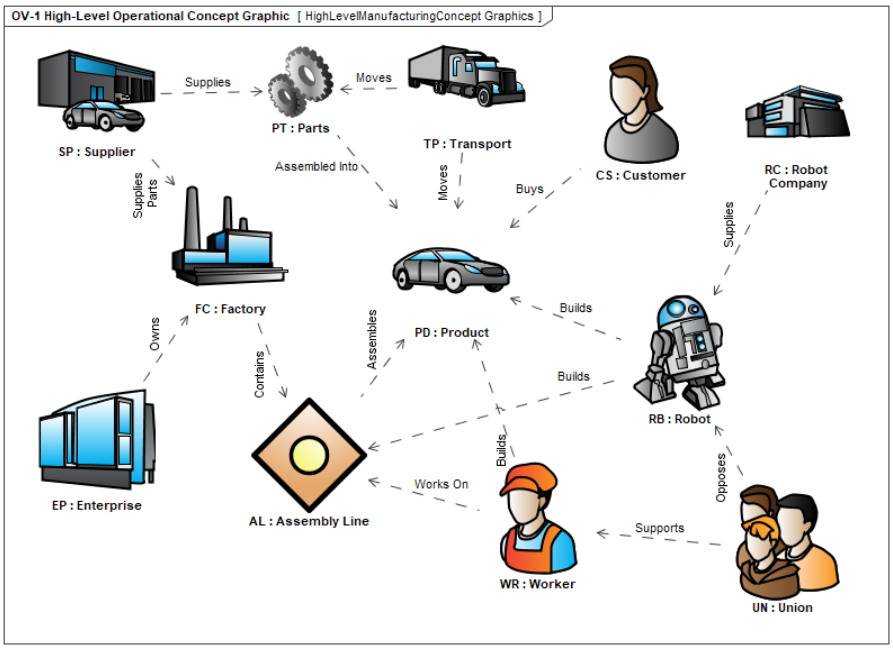

Figure 2. High Level Manufacturing Concept for Powerhouse Engines

Having defined the main enterprise concepts, a set of capabilities for the enterprise has been developed as shown in Figure 3. Capabilities are also logical entities which define what is desired without defining how. A capability is the ability to achieve a desired effect realized through a combination of ways and means (e.g., systems and other elements in the resources view) along with specified measures.

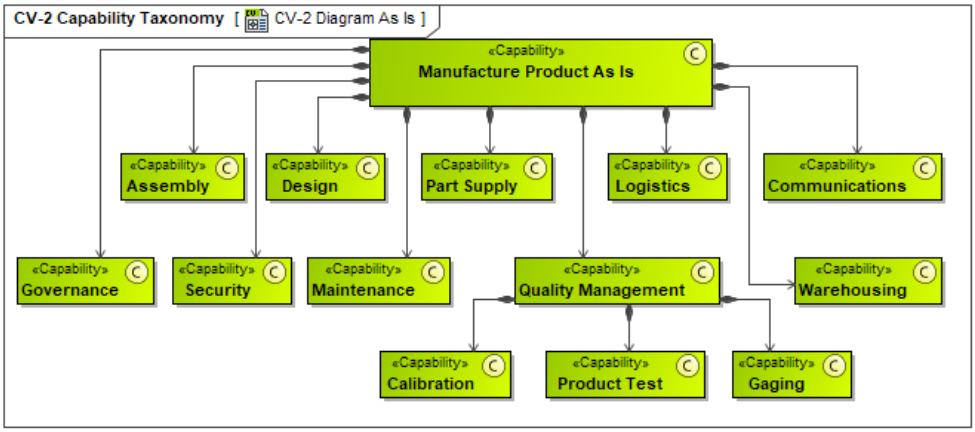

Figure 3. Powerhouse Engines Enterprise Capabilities

The main capability for the as-is enterprise shown in Figure 3 is to manufacture products. The different phases will deploy additional capabilities. The part supply capability (described above as part supplier) will be implemented by a variety of different resources. Other capabilities such as assembly, design, logistics, etc. are also shown. The Assembly capability can be implemented by a combination of people, tools, robots, automation, robots, etc. These capabilities will be valid throughout the different temporal phases of the enterprise. These capabilities are then further explored by mapping operational activities as shown in Figure 4.

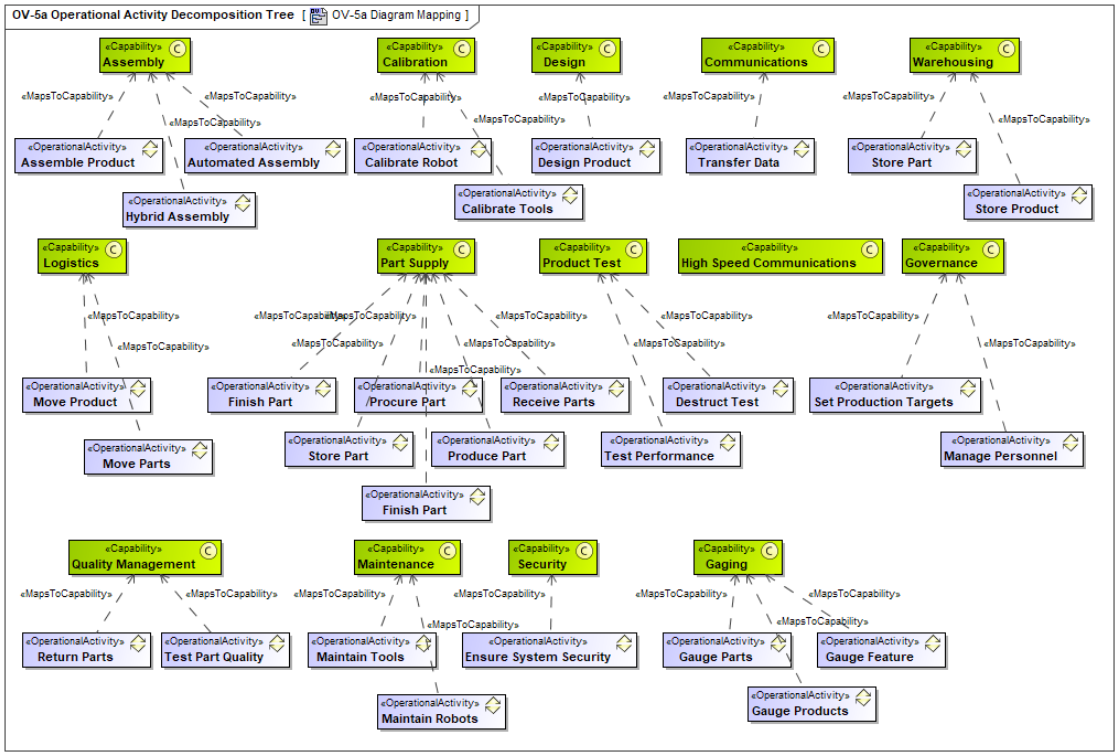

Figure 4. Capability Mapping to Operational Activities

Once again, the elements in the operational view are shown as solution independent. For example, produce part could be provided by any of the three suppliers listed previously. The products are procured based on a design and then are produced for assembly. Ensure system security can be implemented via a combination of people, systems, cameras, sensors, AI, security gates and fencing, etc. Much of it can also be outsourced to an external supplier by defining appropriate services. These operational activities are then grouped in a logical order to show the manufacture process as shown in Figure 5.

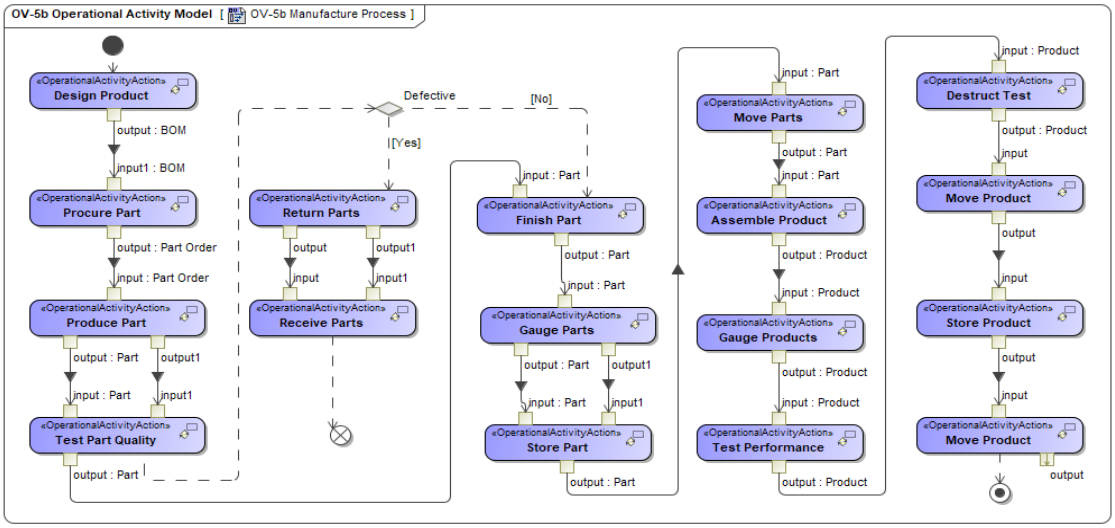

Figure 5. Manufacture Process Operational Activity Diagram

Defining the activities and the order in which they occur, as well as the inputs and outputs helps define the main behaviors in the system, data exchanges as well as physical interactions and interfaces. Error conditions as well as alternate routes can also be defined, as well as performance measurements and constraints. The operational activities can be grouped together by purpose to help define the operational performers for the logical architecture. An example of this is shown in Figure 6.

Preventing "Solutioneering" Rather than Engineering

"Solutioneering" means attempting to solve a problem or deal with a difficult situation without understanding it or starting out with a solution and building the system around it. The logical architecture helps prevent this. The operational performers are created based on the logical, causal, temporal, physical and other groupings of the defined activities. The operational activities define what needs to be done and the operational performers execute those activities. The interactions shown on the diagram are linked to the interactions on the activity diagram. This ensures that the proposed functional interactions can be implemented using the structural components. Having done this, the activity diagram can be updated with swimlanes corresponding to the structural elements. This performs an explicit allocation of the activities to the operational performers.

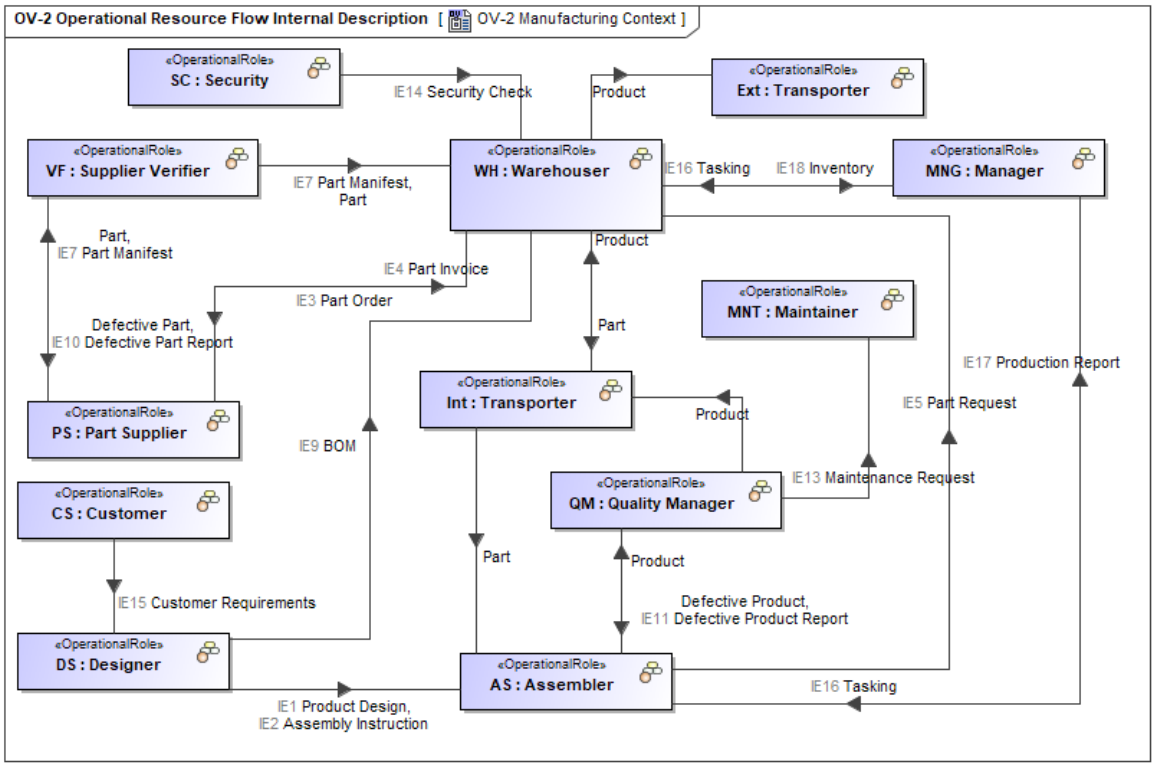

Figure 6. Manufacturing Operational Structure

Also note that the exchanges are non-specific. For example, a product is assembled using parts rather than a specific type of engine made of specific parts. This provides maximum flexibility in the development of the solution architecture. For example, the current organization manufactures internal combustion engines mostly using manual processes. Future plans are to manufacture electric engines using modern means of production such as robotic systems, AI, and parts created

using 3D printing. This operational architecture will support both solution architectures as well as many others providing maximum flexibility. Figure 7 shows the mapping between the operational performers and the implementing systems for the as-is architecture. This traceability allocates responsibility for the logical onto the physical systems implying that the defined logical interactions and behavior will be implemented by the physical elements.

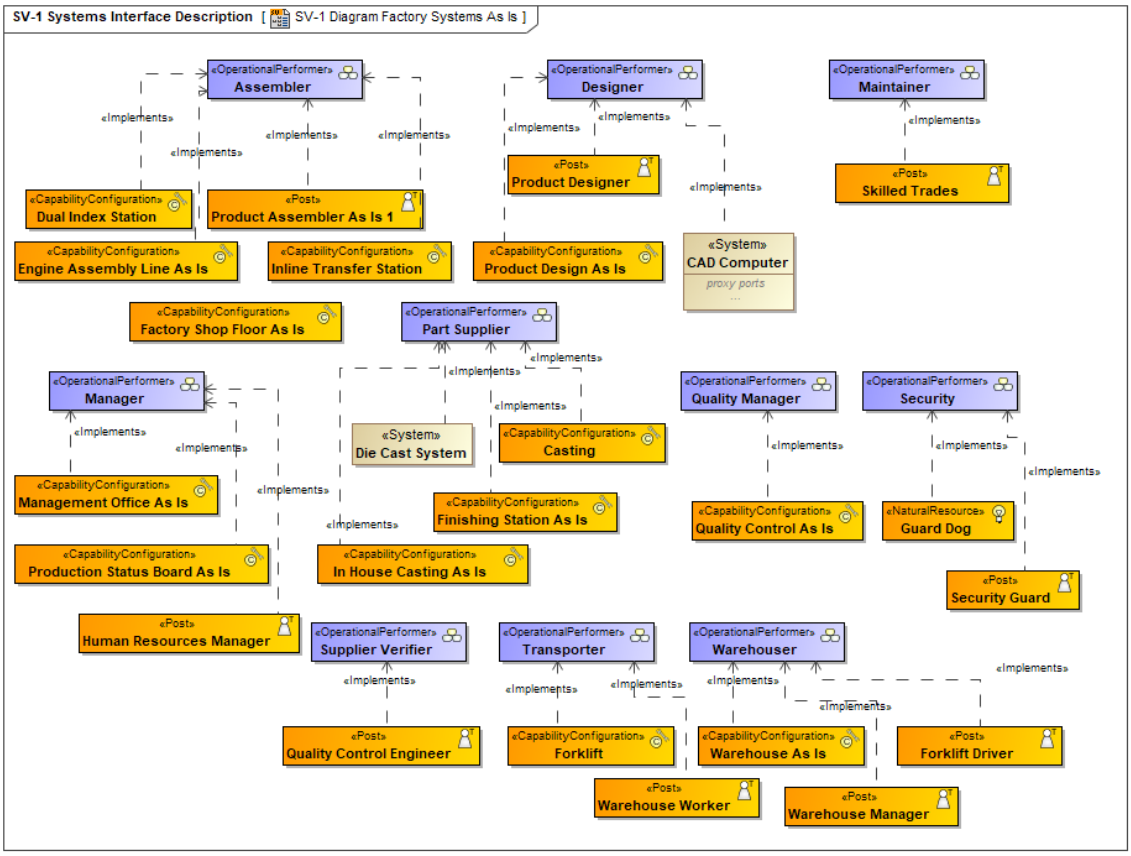

Figure 7: Mapping Between Operational Performers and Systems

Part Supplier is mapped to the diecast system, casting, and finishing station. In future phases this is mapped to an outsourced service, and internally provided 3D printers. Other mappings are to people, systems, software, etc. The interactions, functions, data, etc., can also be mapped as well and traceability tables generated. The purpose of the As-Is operational and resource architecture and mapping is to define the logical architecture that will define the structural and functional elements that will be implemented in future phases. By abstracting the physical to the logical, the true purpose of the existing systems can be specified from which new and innovative solutions and architectures can be defined.

Providing business objects for service definition

The relationship between architecture data elements across the Operational Viewpoint to the Service Viewpoint and Capability Viewpoint can be exemplified as services are procured and fielded to support organizations and their operations or a capability. (MODAF, 2020) Services in UAF are intended to allow the operational layer to be developed without impacting on the resource layer provided that the operational layer only makes use of the functionality provided by the ser-

vice interfaces. In the same sense the resource layer can also develop on its own without impacting on the operational layer provided that the service interfaces are untouched. (Hause, Kihlström, 2021) NAF version 4 places services directly underneath the strategic layer and assumes that services are created directly from strategic decisions. (NATO, 2020). Figure 8 shows the mapping between the Part Supply capability, the produce part operational activity and the casting service.

The Casting service specification exhibits the part supply capability, in that it will realize its required elements. The operational activity Produce Part "consumes" this service. Consumes will be renamed to implements in UAF version 1.2 and the direction of the relationship reversed. In addition, the implementing systems of the services can also be shown with multiple system implementation possibilities. Performance attributes in terms of KPIs and others are defined for potential service providers. For more on the services views see (Hause, Kihlström, 2021)

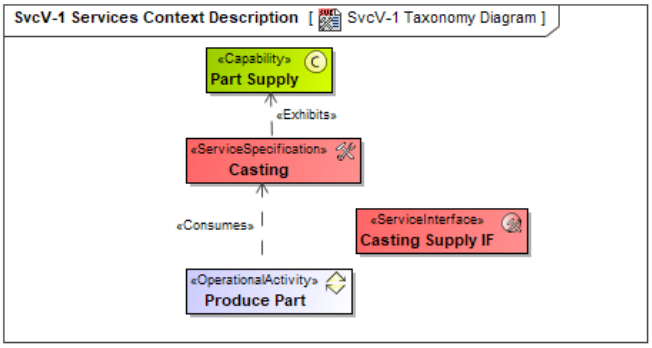

Figure 8. Service Mapping Between Capabilities and Operational Elements

Providing a Bridge from Requirements to the Solution

The aim of the approach is to progress from system requirements (representing the problem from a supplier/designer point of view, as independent of technology as possible) through an intermediate model of logical architecture to allocate the elements of the logical architecture model to system elements of candidate physical architecture models. System requirements and logical architecture models share many characteristics, as they are both organized on functional lines, independently of the implementation. Some authors (Stevens et al. 1998) go so far as to conflate the two, which simplifies the handling of multiple simultaneous views.

Design decisions and technological solutions are selected according to performance criteria and non-functional requirements, such as operational conditions and life cycle constraints (e.g., environmental conditions, maintenance constraints, realization constraints, etc.). Creating intermediate models, such as logical architecture models, facilitates the validation of functional, behavioral, and temporal properties of the system against the system requirements that have no major technological influence impacts during the life of the system, the physical interfaces, or the technological layer without completely questioning the logical functioning of the system. (INCOSE, 2021)

Bridging the Gap Between Capabilities and Implementation

Figure 9 shows the mapping between the defined capabilities, operational activities that map to them, and the resource functions performed by the systems, software, personnel, etc. that imple-

ment the operational activities. This behavioral mapping shows the traceability from the capabilities down to the implementing functions. A similar structural mapping can also be created that traces from the capabilities to the operational performers to the implementing resources. At face value, these traceability charts demonstrate that the required capabilities have all been implemented. It also shows the "what" and "how" of this mapping across the different layers.

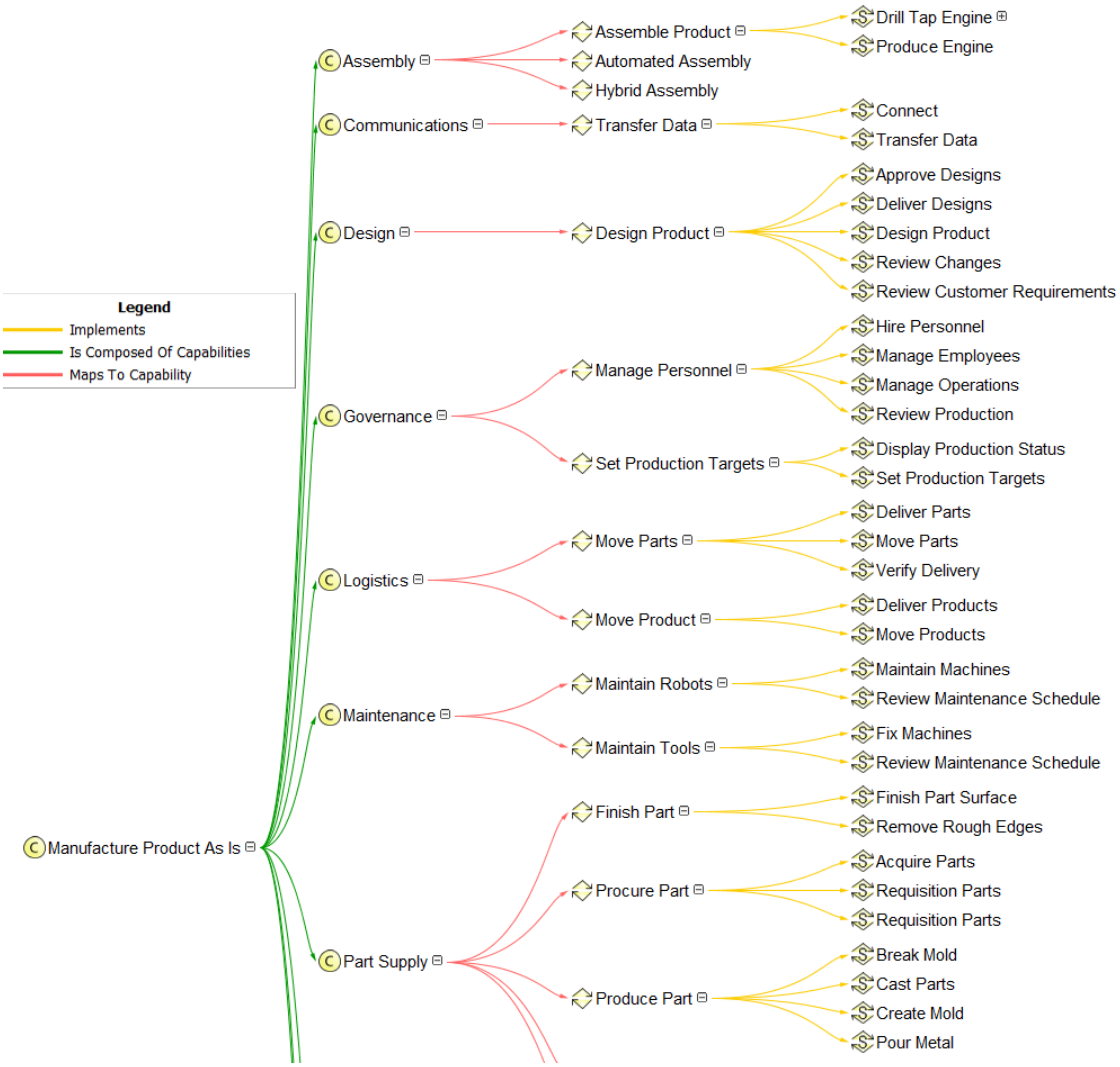

Figure 9: Enterprise Behavioral Mapping

Logical Architectures Help Spur Innovation

Systems engineering involves translating customer needs into viable systems that meet those needs. All systems over time become dated, competitors introduce faster, better, and cheaper products that can reduce demand, technologies will evolve, and systems environments can change. Therefore, it is necessary for companies to occasionally rethink their products, systems and solutions to reimagine new systems. The iPhone is one example where several devices in our pockets were combined to provide a completely new product. In the 1950s and 1960s containerized ship-

ping was developed by first conceiving and manufacturing reusable, standardized, containers and then special ships for carrying them. These two innovations have greatly increased shipping capacity and reduced costs substantially.

Finally, the Tesla electric car changed the configuration of the engine. In automobiles, the purpose of the engine is to provide torque to drive the wheels. Rather than creating a single engine and drive shaft, Tesla vehicles have individual engines for each wheel. This improves handling, reduces cost, reduces vehicle weight, and removes links in the chain from power source to the target device. This is only possible by looking at the functionality and purpose of a system and its elements in a solution independent way and imagining the number of ways in which they could be realized. This can lead to large leaps of innovation rather than merely incremental improvements.

Physical Elements in the Logical Architecture

That said, it needs to be remembered that a logical architecture that avoids solutioneering will need to deal with boundary conditions as well as the context within which the logical architecture of interest finds itself. These boundary conditions are dealt with explicitly in the architecture frameworks such as UAF and show up there as Known Resource elements in the model. This means that logically they will need to be handled by the elements that are purely logical and will act as restrictions on the logical behavior allowed. Known resources may be things that have been designed by someone else or purely natural elements.

A logical transportation element in an architecture dealing with the management of municipal transportation through a city will need to deal with known resources such as roads, traffic lights and speed limits, transportation lanes etc. based on the overall constraint that those are the elements that a municipal transportation element needs to deal with. If the logical architecture becomes “too fluffy” and avoids dealing with such things one can end up with a logical architecture that is unusable since it is too far removed from something concrete. A useful metaphor when creating a logical architecture is to consider oneself as an observer in a balloon looking down onto the elements and determining the logical reason for their existence by observing what they can do, taking all constraints and logical requirements into account. When designing, that viewpoint is reversed, and the designer is situated within the elements and observing the world by looking out. However, what this implies is that the creation of a logical architecture is a balancing act.

As an example, take the case of an electric-powered construction site [Sjoberg, et al, 2017]. The key was a logical architecture that lived with the constraints imposed by the requirements which can be summarized as transport crushed material from a place where it was produced by enabling said material to be loaded and transported through the site in an efficient manner. Several known resources constrained the architecture: the roads throughout the site, topography, weather, material production speed, loading ability, transportation ability with the logical elements being subdivided into loaders and transporters with the additional constraint that they be electrical and autonomous. The logical architecture was constrained by these elements and any attempt to disregard them would cause the logical architecture to become useless. Restrictions of the logical elements is sometimes considered as “solutioneering” but in this case is rather the result of object-oriented analysis and not design.

Logical architectures and their use in a SoS

The manufacturing situation described previously is essentially a system of systems. A crucial point in a system of systems is that the elements within it have been developed with separate life cycles, i.e., the scenario that describes the system of systems has not been considered during the basic development of the systems that are being used within the scenario. There may be additions that have been introduced into an otherwise known resource to enable the operations within the scenario, but this is a small addition to an otherwise known resource. In such a system of systems scenario all the components may essentially to some degree be known resources.

The logical model for such a system of systems can be used to analyze hazards that are the result of failures in the logical interaction between elements that the scenario requires. An example of a logical model of this kind is seen below. It is particularly useful for this scenario as one of the proposed technologies is autonomous vehicles to select and deliver parts throughout the factory. The scenario is called platooning and describes a situation where a set of trucks for a convoy with very small distances in between the trucks to mitigate congestion and to reduce fuel consumption (for all but the lead truck) as shown in Figure 10.

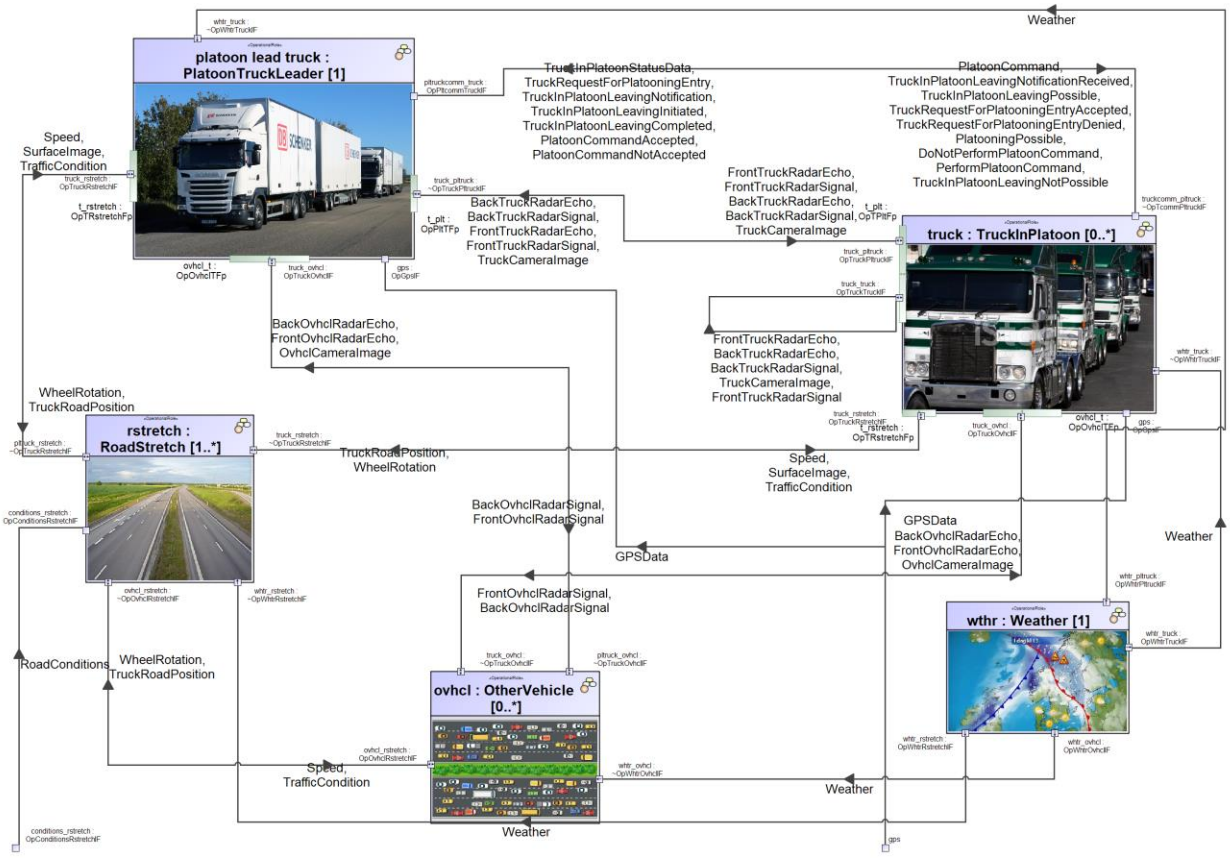

Figure 10: Truck Platooning Operational Connectivity

The above depicts the overall scenario for platooning and recognizes that the leader of the platoon is somewhat different from the rest, something that the analysis of the model and the logical interactions required to make the platoon work quickly demonstrated.

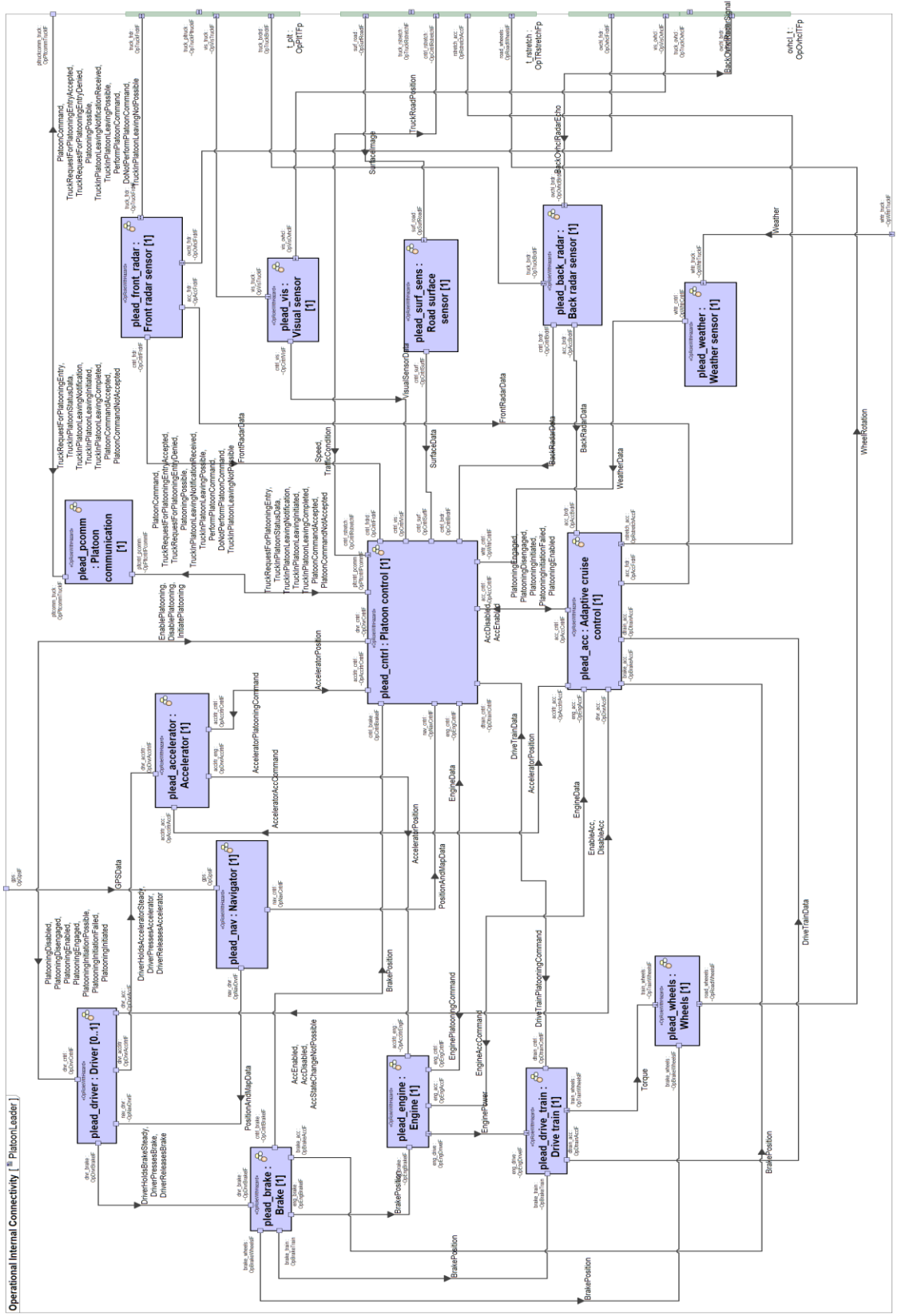

Figure 11: Truck Platooning Logical Interactions

To continue the analysis of potential hazards, the platoon leader as well as the trucks in the platoon had to be broken down one step further into a set of logical components that by and large are known resources with some modifications and additions based on the platoon requirements. The breakdown of the platoon leader is shown in the Figure 10. The key addition here is the platoon control part. A logical breakdown of the truck in platoon shows essentially the same components with some changes to what the platoon control is responsible for implying that any truck, by changing its behavior as far as the platoon control is concerned can act as the platoon leader. By considering both the known components as well as any additional platooning features or functions within the trucks and the interaction with all of the external known resources as well as internal known resources (driver, brake, accelerator etc.) hazard scenarios can be analyzed that are the result of the logic determined by the scenario rather than individual design. Figure 11 shows examples of such hazards as examples of the analysis possibilities based on the logical model.The individual hazard scenarios can be further explored by means of a sequence diagram as shown in Figure 12

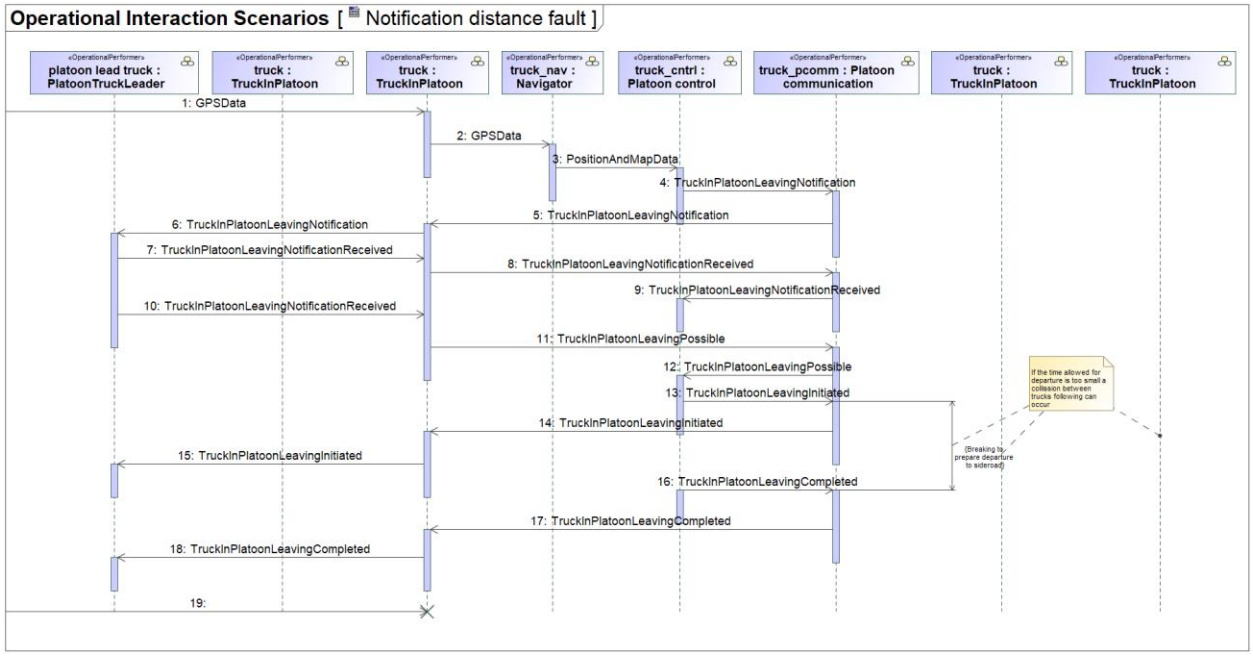

Figure 12: Truck Platooning Hazard Scenario

- If a truck somewhere in the platoon wishes to leave by exiting to a side road, the platoon knowledge of the capabilities of the trucks within needs to be such that when the truck that is to leave starts braking, the trucks behind can cope with this. This is essentially a platoon information handling issue that has to do with the knowledge of the performance associated with the trucks in the platoon.
- Any change that the lead truck communicates to the members of the platoon needs to be handled such that all trucks can react without problems.
- Several normal features within a truck that participates in a platoon needs to be disabled such as, in most circumstances, braking and accelerating performed by the driver as well as the individual adaptive cruise control of the trucks that participate in the platoon.

An interesting point to note here is that the hazard shown here is not the result of an error in any equipment but rather the result of incorrect operational information in the platoon as such,

something that a logical model can detect more easily than a completed detailed resource model where this interaction issue could easily be overlooked. Performance characteristics of the system can be explored during simulation of the logical system to specify requirements for the physical architecture, to further identify conformant physical systems.

Let's Get Physical

So far, we have defined the solution elements as systems that will be further specified. Most people think of physical architecture as things that can be touched, have geometry details, colors, material, etc. The Actual Resources domain in the UAF define instances of elements defined in the resource's views and specific values of the defined attributes are defined. They are meant to represent things that exist in space and time. However, the UAF resources views do not contain enough information to allow an engineering team to build the system. Further elaboration along the digital engineering thread from concept to physical implementation must be performed in order to specify the details of a system that can exist in space and time. For the next link in the digital thread, systems engineers typically use the Systems Modeling Language (SysML) as a means of modeling their systems. The SysML design can co-exist with the UAF package structure. Traceability can be created using allocations, trace, or other SysML dependencies. Models created in SysML include functional views such as use cases, state and activity diagram, and sequence diagrams. System structure is defined using block definition diagrams and internal block diagrams. These define the various components and assembles of the system and system components. This includes details of the different components or parts in the system, the number of each part, and where it is included in the hierarchy. Other information modeled in value properties can include size, weight, power, cost, etc. This provides the basis for the Bill of Materials or BOM in Product Lifecycle Management (PLM). A BOM or product structure is a list of the raw materials, sub-assemblies, intermediate assemblies, sub-components, parts, and the quantities of each needed to manufacture a physical product. A BOM may be used for communication between manufacturing partners or confined to a single manufacturing plant. Capabilities of an integration between MBSE and PLM have been implemented by multiple tool vendors and third party suppliers and includes:

- Auto-generation of PLM parts from Model System Blocks
- Auto-generation of PLM Options & Variants from Model Variation Points & Variants enabling Integrated product Line Engineering
- Bi-directional traceability to manage traceability links between PLM parts and Model System Blocks
- Reverse engineering of SysML block structure from PLM BOMs.

This part of the digital thread increases productivity by accelerating PLM product/project start-up using pre-populated BoMs & Options. It improves product quality by avoiding the re-entry of data throughout the system & product lifecycle, enabling early impact analysis of system design changes and product part changes. PLM systems can then provide manufacturing instructions and detailed CAD drawings to machines on the shop floor that will result in the final physical system which you can in fact touch.

Conclusion

The logical architecture is essential for concentrating on the major aspects of the system without sliding into solutioneering or starting with too many preconceived solutions. By limiting the initial models and analysis to the functional purpose of the model, the mind of the innovative engineer can explore multiple means of achieving this functionality. This is especially important in complex systems of systems as well as competitive environments where multiple solutions could exist. The argument that "We have always done it this way" can often constrain the creative engineering process and prevent innovation. Logical architectures can help to eliminate this mindset by starting with a solution independent point of view before designing the physically realized system. Of course, creating logical structures is not a skill that all possess. Much of the content in this paper has been guided by the two authors combined 90 years of experience developing systems as well as the sources of systems engineering best practice cited in this paper. There are engineers that see no benefit in spending time on anything other than the eventual solution to be implemented and we have met them. This includes hardware, software, mechanical, chemical, procedural, etc. elements and aspects of a system. "Why are we wasting our time with requirements and analysis when we should be writing code/ banging metal/ ordering parts/ etc." has been the complaint of bad engineers and managers for as long as we have been building systems. Physical implementation and its resulting constraints always need to be kept in mind but limiting our thinking to existing solutions limits us to incremental improvements rather than leaps in progress. Innovators and implementors are both essential to the development of system. Logical architectures that never take form are of no use to anyone. A combination of both is what is required for dreams to turn into reality and great ideas into the systems of tomorrow and of course, things we can touch.

References

DoDAF DoD CIO, 2012, DoD Architecture Framework Version 2.02, DoD Deputy Chief Information Officer, Available online at http://dodcio.defense.gov/dodaf20/dodaf20_pes.aspx, accessed June, 2014.

Bruce Powel Douglass Ph.D., in Real-Time UML Workshop for Embedded Systems (Second Edition), 2014

Hause, M. 2014. "SOS for SoS: A New Paradigm for System of Systems Modeling." Paper presented at the IEEE, AIAA Aerospace Conference, Big Sky, US-MT, 1-8 March.

Hause, M., F. Dandashi, 2015. "UAF for System of Systems Modeling , Systems Conference (SysCon)." Paper Presented at the 9th Annual IEEE Systems Conference, Vancouver, CA-BC, 13-16 April.

Hause, M., Kihlström, L., 2021, An elaboration of Services Views within the UAF, presented at the 2021 Virtual INCOSE International Symposium.

INCOSE 2015, Systems Engineering Handbook Fourth Edition, Published by Wiley

INCOSE 2021, Systems Engineering Body of Knowledge, SEEBOK

MOD Architectural Framework, Version 1.2, 2020, Office of Public Sector Information, https://www.gov.uk/guidance/mod-architecture-framework/

Martin, J 2020, "Enterprise Architecture Guide for the Unified Architecture Framework (UAF)," presented at the INCOSE International Symposium.

NATO Architecture Framework Version 4, January 2018, Architecture Capability Team Consultation, Command & Control Board

Object Management Group (OMG). 2013. OMG2013-08-04:2013. Unified Profile for DoDAF/MODAF (UPDM) V2.1, http://www.omg.org/spec/UPDM/2.1/PDF
Object Management Group (OMG), 2019. OMG2012-06-01.OMG Systems Modeling Language (OMG SysML™), V1.6, http://www.omg.org/spec/SysML/1.6/PDF/.
Object Management Group (OMG), 2019a, The Unified Architecture Framework, (UAF) Version 1.1, Available from https://www.omg.org/spec/UAF
Object Management Group (OMG), 2022, The Unified Architecture Framework, (UAF) version 1.2, expected date of publication, March 2022.
Stevens, Richard, et al, 1998, Systems Engineer Coping with Complexity, by Prentice Hall

Biography

Lars-Olof Kihlström. Lars-Olof Kihlström is a principal consultant at Syntell AB where he has worked since 2013, primarily in the area of MBSE. He has been a core member of the UAF group within the OMG since its start as the UPDM group. He was involved in the development of NAF as well as MODAF. He has worked with modelling in a variety of domains such as telecommunications, automotive, defence as well as financial systems. He is specifically interested in models that can be used to analyze the behavior of system of systems.

Matthew Hause. Matthew Hause is a principal consultant at SSI, a member of the UAF group, and a member of the OMG SysML specification team. He has been developing multi-national complex systems for almost 40 years as a systems and software engineer. He worked in the power systems industry, command and control systems, process control, SCADA, military systems, and many other areas. His role at SSI includes consulting, mentoring, standards development, specification of the UAF profile and training.