Points Of View and Reuse in Logical and Physical Architecture

Lars-Olof Kihlström
CAG Syntell
P.O.Box 10022
10055 Stockholm, Sweden
lars.olof.kihlstrom@cag.se

Matthew Hause
SSI
3208 Misty Oaks Way
Round Rock, Texas, USA
mhause@systemxi.com

Abstract. The terms logical and physical architecture descriptions have been used for a long time. There seems however to be different ideas of what they mean and what they imply. This paper makes use of several examples to illustrate how logical and physical architecture can be used and the advantages that this can provide, especially in a system of system (SoS) context. It discusses how a logical representation, and a physical representation should be viewed i.e., from where the modeler views the system. The issue of so-called solutioneering in a logical architecture model is considered. It also provides a description of the reuse possibilities that exist between the logical architecture and the physical architecture. The examples used to illustrate the points made are all based on the use of the Unified Architecture Framework (UAF) since this explicitly makes the distinction between logical and physical architectures.

Keywords. Logical Architecture, Physical architecture, UAF, Reuse.

Introduction

There is a lot of information concerning the concepts of logical architecture and physical architecture. An attempt at using Google to find information yields many possible sites that discuss the concept in various domains. One definition that turns up dealing with both concepts is the following: “Logical architecture is the decomposition of the system into functional components/services independent of the platform that will execute them, whereas deployment architecture specifies the representation of the logical components in terms of platform-specific entities.” (Farcas et al, 2014). It is assumed here that deployment architecture implies the same thing as physical architecture. Sometimes one gets the feeling that the concepts are essentially standalone and that the relationship in between them is not explicitly defined.

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 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." "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) Physical models will also contain these as well as traceability to the logical model elements.

Chapter 16 of Friedenthal et al (2011) describes the logical architecture definition using the Object-Oriented Systems Engineering Methodology (OOSEM). "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 et al, 2011)

Having a well-defined logical architecture is an essential part of the development process as it forms the "requirements set of views" upon which the rest of the development process depends. Traceability to this set of views is essential as it justifies the inclusion or exclusion of elements from the physical architecture based on the ability to trace to the logical model.

ISO 42010 Views, Viewpoints, etc.

Prior to looking at the main concepts in this paper, it is important to define some commonly used terms in the regarding architecture as they differ from normal English use. The majority of the following definitions are taken from the SO/IEC/IEEE 42010:2022 Software, systems and enterprise Architecture description standard.

- Enterprise – a "human undertaking or venture that has a mission, goals and objectives to offer products or services, or to achieve a desired project outcome or business outcome" (ISO 42010 2021).
- Architecture – fundamental concepts or properties related to an entity in its environment and governing principles for the realization and evolution of this entity and its related life cycle processes (ibid).
- Enterprise architecture – a set of "fundamental concepts and properties … and governing principles for the realization and evolution" of the enterprise (ibid).
- Architectural Description – a work product used to express the Architecture of some System of Interest (ibid). OMG (2022) adds that it "provides executive-level summary information about the architecture description in a consistent form to allow quick reference and comparison between architecture descriptions -- It includes assumptions, constraints, and limitations that may affect high-level decisions relating to an architecture-based work program."
- Concern –a "matter of relevance or importance to a stakeholder regarding an entity of interest" (ISO 42010) that will be addressed in an architecture).
- View –expresses the architecture of the system-of-interest in accordance with an architecture viewpoint (or simply, viewpoint). (Note: a view is an "information item, governed by an architecture viewpoint, comprising part of an architecture description" (ISO 42010) that communicates some aspect of an architecture and expressing the architecture from the perspective of specific stakeholders regarding specific aspects of the architecture entity and its environment (ISO 42020)).
- Viewpoint –frames (to formulate or construct in a particular style or language) one or more concerns. A concern can be framed by more than one viewpoint. (Note: a viewpoint is a "convention for the creation, interpretation and use of an architecture view to frame one or more concerns" (ISO 42010) that governs the creation of views).

- View Specification – The representation of a view from a specific viewpoint. In the context of the grids in the following section, the View Specifications at the boxes at the intersections of the rows and columns.

A Tale of Several Architecture Frameworks

The related architecture frameworks described below were designed to make it easier to describe architecture for an enterprise as well as a system and system of systems. They all deal with the distinctions between the logical and the physical architecture to various degrees. In the 1990's the US Department of Defense started with an existing framework used for Command and control systems named C2ISR and created DoDAF Version 1.0 (Department of Defense Architecture Framework). A summary of the viewpoints defined for DoDAF can be found in Figure 1.

– All views:	Adminstration, commonly used elements
– Operational views:	Business operations
– System views:	Technology realizations
– Standard views:	Technical and operational standards

Figure 1. DoDAF Viewpoints Version 1.

Each viewpoint contained a set of defined views, each describing a given set of information regarding the system of interest to be documented as part of the view. The UK found areas that their Ministry of Defence (MOD) wanted covered by an architecture framework, so their MOD created MODAF (Ministry Of Defence Architecture Framework). The additional viewpoints are shown in Figure 2

– All (AV):	Adminstration, commonly used elements AV-1 through to AV-2
– Strategic (StV):	Overall strategic directions StV-1 though to StV-6
– Operational (OV):	Business operations OV-1 through to OV-7
– Services (SOV):	Service specifications SOV-1 through to SOV-5
– Systems (SV):	Business realized by technology and people SV-1 through to SV-12
– Standards (TV):	Technical and operational standards TV-1 through to TV-3
– Acquisitions (AcV):	Development programs AcV-1 through to AcV-2

Figure 2. MODAF Viewpoints.

NATO created an architecture framework named NAF version 3.0 (NATO architecture framework) that was based on MODAF. A partial new version in the form of NAF 3.1 was also created based on updates to MODAF. The Service was created initially as part of the work in NATO. It was adopted in the later versions of MODAF. The service viewpoint was eventually introduced in DoDAF, both in DoDAF version 1.5 and DoDAF version 2. Based on the appearance of similar but different frameworks

that were not being updated synchronously, the Object Management Group (OMG) started to develop an architecture standard that combined both MODAF, NAF and DoDAF to be used by tool vendors. It was named UPDM (Unified Profile for DoDAF and MODAF). As time went by NATO created a new version of NAF in the form of NAF version 4 and the grid that defines viewpoints and views. The different views can be seen in more detail can be seen in **Fel! Hittar inte referenskälla.**Figure 3.

The NAF row named Logical Specifications deals with the Logical Architecture and the row named Physical Resource Specifications deals with the Physical Architecture. Within the NAF version 4 specification, the logical viewpoints row is defined as follows: "The Viewpoints in the Logical Specifications row of the NAF grid support the solution-independent description of the logical nodes (elements of capability), activities, and resource/information exchanges required to accomplish missions." Within the NAF version 4 specification, the physical resource viewpoints row is defined as follows:

"Viewpoints in the Physical Resource Specifications row of the NAF grid support the description of the structure, connectivity and behavior of the various types of Resources. Resource Types include people, organizations, artefacts, software and configurations of any or all of them. In particular, these Viewpoints are used to specify how Types of Resources are configured and connected to deliver Capabilities and Services. "The relationship in between the logical and physical specifications are dealt with by The L4-P4 viewpoint. "The L4-P4 Viewpoint is concerned with the linkage between functions described in P4, Resource Functions, and operational activities specified in L4, Logical Activities."

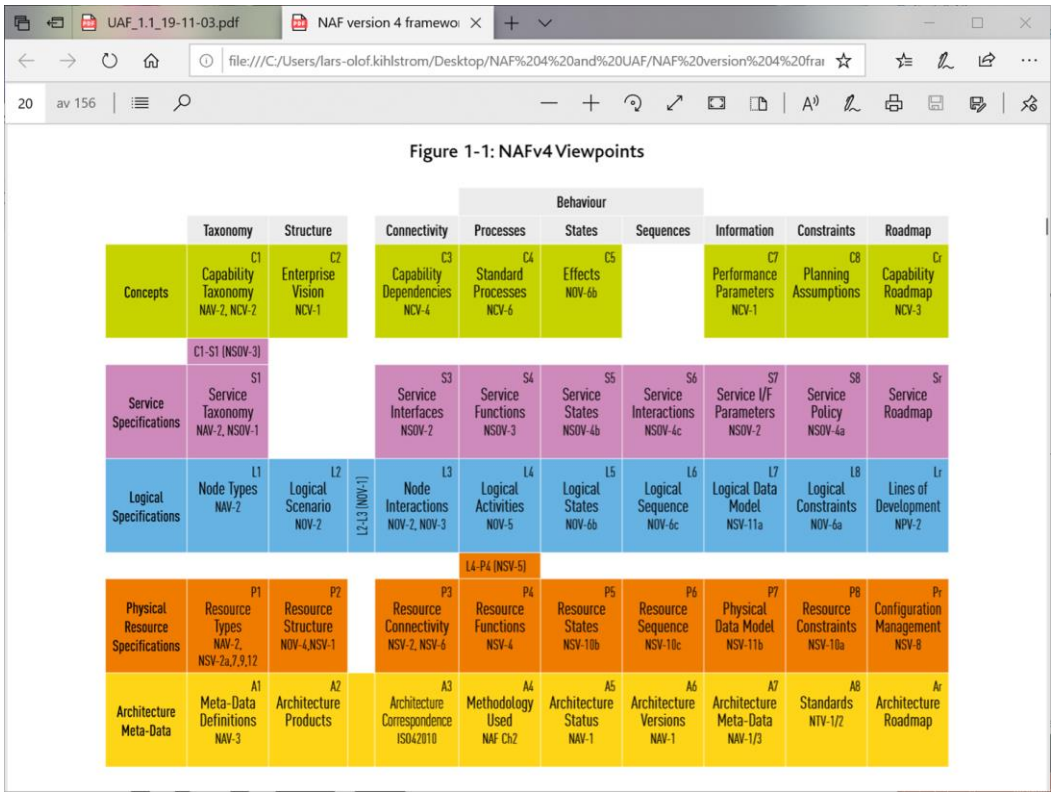

Figure 3. NATO Architecture Framework Version 4 (NAF V4)

Additional frameworks appeared dealing with security, human-factors as well as systems of systems. These developments were reviewed by the OMG as new versions of UPDM was created. With the additional concerns that needed to be covered, UPDM had outgrown the name and the third version of UPDM was named UAF (Unified Architecture Framework). UAF created a set of viewpoints taking a number of issues int account as shown in the viewpoint explanation in Figure 4. The detailed grid describing the views in each viewpoint for UAF version 1.1 is shown in Figure 5. Since this was published,

UAF has been updated once. The viewpoints dealt with remain the same but some changes to the views have been performed. The detailed grid for UAF version 1.2 is shown in Figure 6.

Work is ongoing within OMG for a new update which will describe a common meta-model and two implementations one making use of SysML version 1.7 and one making use of SysML version 2.

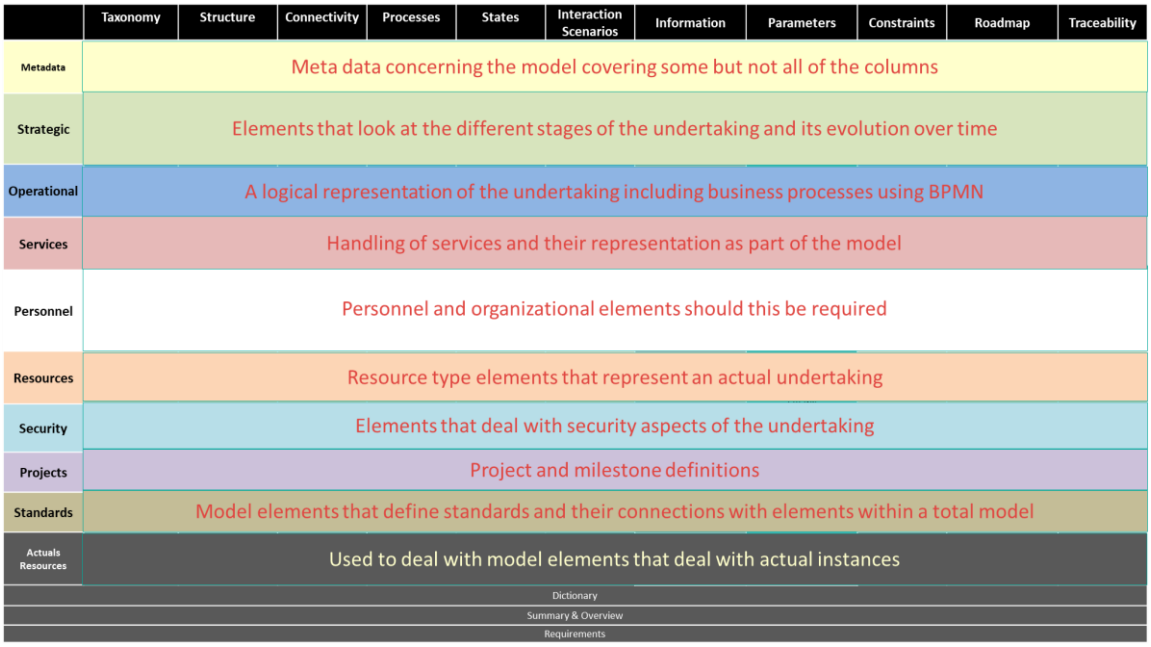

Figure 4. Unified Architecture Framework (UAF) viewpoints

	Taxonomy	Structure	Connectivity	Processes	States	Interaction Scenarios	Information	Parameters	Constraints	Roadmap	Traceability
Metadata	Metadata Taxonomy	Architecture Viewpoints	Metadata Connectivity	Metadata Processes	-	-	Conceptual Data Model, Logical Data Model, Physical schema, real world results	Environment, Measurements	Metadata Constraints		Metadata Traceability
Strategic	Strategic Taxonomy	Strategic Structure	Strategic Connectivity	-	Strategic States	-			Strategic Constraints	Strategic Deployment, Strategic Phasing	Strategic Traceability
Operational	Operational Taxonomy	Operational Structure	Operational Connectivity	Operational Processes	Operational States	Operational Interaction Scenarios			Operational Constraints	-	Operational Traceability
Services	Service Taxonomy	Service Structure	Service Connectivity	Service Processes	Service States	Service Interaction Scenarios			Service Constraints	Service Roadmap	Service Traceability
Personnel	Personnel Taxonomy	Personnel Structure	Personnel Connectivity	Personnel Processes	Personnel States	Personnel Interaction Scenarios			Competence, Drivers, Performance	Personnel Availability, Personnel Evolution, Personnel Forecast	Personnel Traceability
Resources	Resource Taxonomy	Resource Structure	Resource Connectivity	Resource Processes	Resource States	Resource Interaction Scenarios			Resource Constraints	Resource evolution, Resource forecast	Resource Traceability
Security	Security Taxonomy	Security Structure	Security Connectivity	Security Processes	-	-			Security Constraints	-	Security Traceability
Projects	Project Taxonomy	Project Structure	Project Connectivity	-	-	-			-	Project Roadmap	Project Traceability
Standards	Standard Taxonomy	Standards Structure	-	-	-	-			-	Standards Roadmap	Standards Traceability
Actuals Resources		Actual Resources Structure,	Actual Resources Connectivity,	Simulation					Parametric Execution/ Evaluation	-	-
Dictionary											
Summary & Overview											
Requirements											

Figure 5. UAF version 1.1 grid

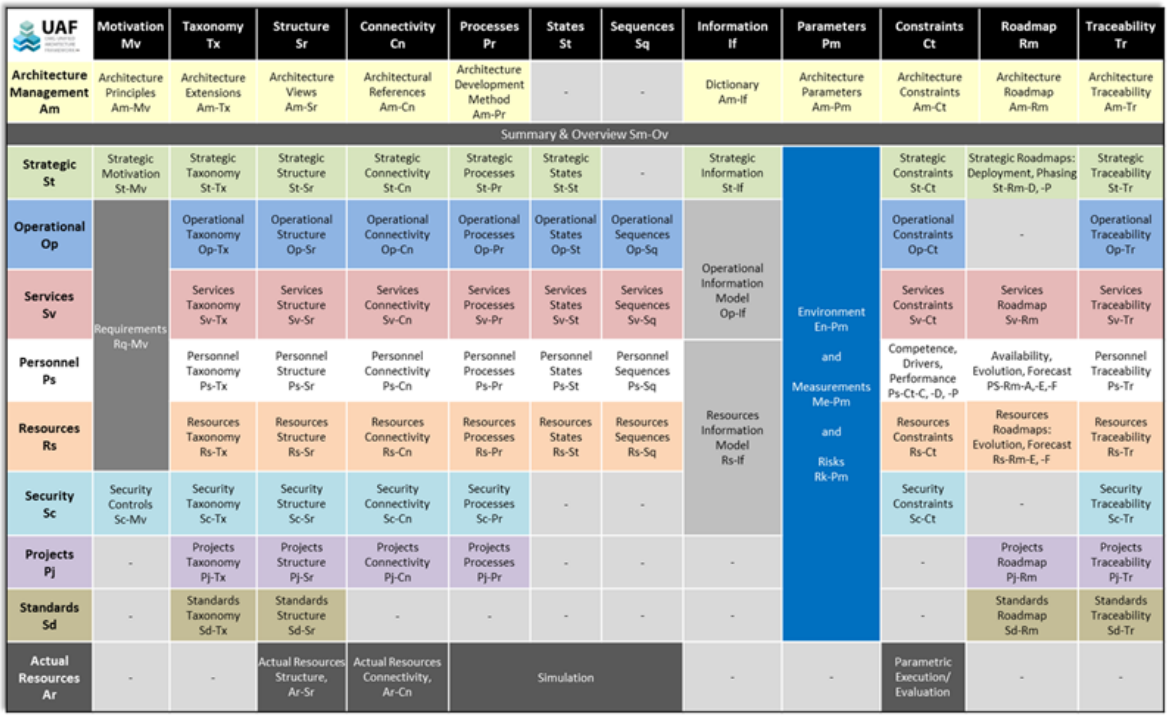

Figure 6. Unified Architecture Framework (UAF) version 1.2

As can be seen the grids shown in Figure 5 and Figure 6 are slightly larger than the one shown in Figure 3, that is; they contains more rows and columns. This means that an architecture model using the grid in Figure 5 or Figure 6 can contain additional information compared to the one shown in Figure 3. The grid in Figure 3 can be mapped to the grids in Figure 5 and Figure 6, i.e., all the information contained in the smaller grid can be visualized within the larger grids.

Within the UAF specification, the Operational view specifications are defined as follows: “Definition: describe the requirements, operational behavior, structure, and exchanges required to support (exhibit) capabilities. Defines all operational elements in an implementation/solution independent manner.”

Within the Unified Architecture Framework specifications, the Resource view specifications are defined as follows: “Definition: captures a solution architecture consisting of resources, e.g., organizational, software, artifacts, capability configurations, natural resources that implement the operational requirements.”

The traceability views in both the operational and the resource domain essentially says the same thing as stated in NAF version 4. The detailed domain meta-model in Figure 7 shows the relationship that is supported between the elements within the resource model and the operational/ logical model.

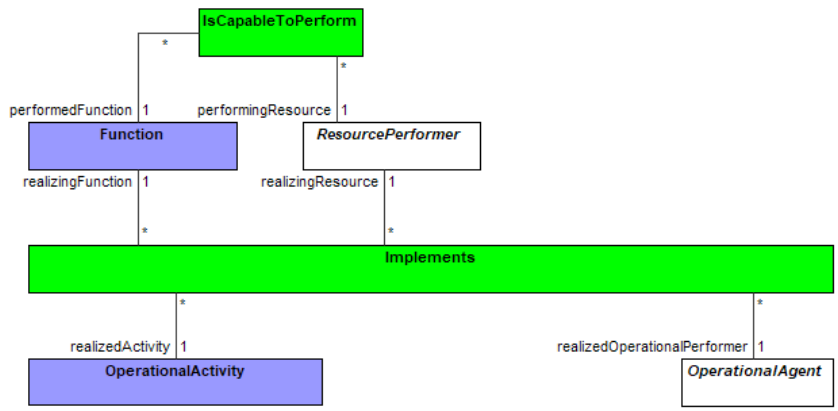

Figure 7. Relationship Definition Between Operational/Logical and Resource Elements in UAF.

Here the green elements represent relationships, and the blue ones represent type element. The white elements shown with the text in italics are abstract, i.e., there are several concrete elements in the framework that inherit from them (Logical/operational: OperationalPerformer, OperationalArchitecture; Resources: ResourceArchitecture, CapabilityConfiguration, System, ResourceArtefact, NaturalResource, OrganizationalResource etc.). The key relationship between the operational/logical elements and the physical resource elements is Implements. An operational agent can be shown as being implemented by a resource performer or resource performers. In the same manner one or more operational agent may be shown as being implemented by a resource performer. One or more operational activities can be shown as being implemented by one or more functions that can be performed by a resource performer. The real semantic meaning of the implements relationship can be quite complex.

Is this all there is? Is there more to this kind of relationship than meets the eye? A possible answer to this question depends to a great extent on how operational/ logical architectures as well as physical resource architectures are viewed and used.

Points Of View as regards logical and Resource Architectures

Persons who have spent time doing both operational/logical architectures as well have physical architectures may have noticed that the way of looking at the architecture elements are different. The following statement can be seen as a way of describing the difference in as short a sentence as possible.

When creating a logical/ operational architecture the logical elements required from a non-implementational perspective are viewed from the outside. When creating a physical architecture the resource elements required from an implementation-specific perspective are viewed from within.

This means that there is a profound change of perspective when dealing with these two kinds of architectures. This is especially visible when attempting to come to grips with system of systems. An operational/logical architecture could be seen as making use of a perspective where the observer is attempting to describe the interactions and behavior of the logical elements when viewed from a balloon 100 meters above the system of systems in question. A similar approach for a logical architecture model can be employed for much smaller systems that are composed of a set of logical components. The observation height there may not be 100 meters but is still from the outside of the elements looking in rather than from inside looking out. The most important aspects are what is visible to the observer in terms of inputs, behavior in response to these inputs, and resulting outputs.

A physical architecture however will have as its starting point an actual implementation perspective of each component and will view the complete system context from within the component being described. A reader might at this point have noticed the term logical components as well as systems within a system of systems and drawn the conclusion that this may well represent a significant departure from a logical view of functionality. They may then have decided that what is being described here is solutioneering i.e., a situation where the logical model is no longer logical but has been infected with solutions.

Creating a logical architecture is a form of balancing act. A perfect logical description of how something is to be done still needs to be implementable in some form. If the logical functions become too amorphous the implementation will end up such that there is no real connection or mapping between the logical/ operational architecture and the physical resource architecture despite any “implements” relationships that have been created between them and between structural elements.

If the operational/ logical architecture is to be of any use it needs to end up as a description that can be used during the complete life cycle of the Enterprise/ System of Systems/ System to analyze and make determinations of the impact of changed contexts, changed external environment, or changed requirements. Once such a logical analysis has been made, the implementational impact can then be analyzed. Avoiding the logical analysis within a usable logical model and starting with the implementation can result in the bigger picture being missed and that the change becomes unnecessarily complicated and time-consuming. Using the logical model to speed things up does however require that the logical and

physical model elements have proper implements traces in between them that can be analyzed, understood, and made use of.

For a logical/ operational model to be useful a couple of concepts need to be properly handled:

- Constraints
- Known resources.

Constraints and Known resources.

When creating a logical/ operational architecture of anything there are always constraints that need to be adhered to as part of the logical elements that make up the architecture. These constraints can be non-functional (weight, size, velocity, processing speed, available technologies) as well as functional in the form of logical requirements that constrain the elements that are to be used within the architecture. These can include conformance to environmental standards, policy in accordance with the domain, or legal regulations due to the geographical location. Implementing a system in Kiruna in Northern Sweden with have different constraints than a system in the Algarve, Portugal. To ignore these constraints even in the early stages of analysis in not wise nor beneficial to economical system development.

To guide the implementation, one must indicate the logical elements within the operational/ logical architecture that are intended to satisfy functional requirements once they have been implemented. The logical elements then need to show how the logic is being fulfilled within the model. The physical resource architecture can use this as a means of determining the detailed functionality to be implemented based on the solution space constraints.

A key issue here is also to ensure that the logical elements show a life cycle perspective. This means that the model elements need to take account of all the logical requirements that they need to be able to deal with. A logical element may well have to do several things in parallel and deal with any interference and interactions between the different actions that are required. Ensuring that this is the case will make the ensuing implementation easier since the implementation then only needs to concentrate on pure implementation issues rather than a combination of logical *and* implementation concerns.

Known Resources

The Known resource concept is a way to deal with boundary conditions. Any System or System of Systems or indeed Enterprise will contain elements or boundaries that it must adhere to, i.e., they are the as they are and must be accepted and interacted with in the way they have been defined. Known resources can be either outside the set of elements under consideration or they can be elements within that cannot be changed in any fashion. Ignoring these elements will result in a system that will not operate within its environment.

A typical example of constraints and known resources is the kind of predetermined platform that has been decided for a set of kinds of products. An example of this would for instance be a platform decided upon for different models of cars marketed by a manufacturer. Each model of the car must make use of the specified generic platform. This implies that each model has a set of known resources that needs to made use of. Each of these platform components is responsible for a given set of logic that need to be used by any additional system or functionality that a specific model of the car needs. The constraints, both non-functional as well as logical, taken together with known resources impose restrictions on what the logical model needs to describe.

When creating logical models that take all the above into account there is sometimes an accusation of solutioneering, i.e., creating a model that presupposes a solution. The previously mentioned balancing act is precisely what is required when weighing the differences between solutioneering and the elements defined for a logical model. Once this balance is achieved, the logical model can be used as a tool for simulations and will directly benefit the actual implementation. The examples shown in the next section are an attempt to illustrate this.

Operational/ Logical Architectures and Physical Architectures

The following examples are elaborations of existing models, papers and standards published previously. The electric quarry example is taken from the Sjöberg et al (2017) paper, entitled An Industrial Example of Using Enterprise Architecture to Speed Up Systems Development. It described a model created for the quarry to define the necessary configuration of chargers, autonomous vehicles, etc. for the quarry as well as cost considerations. In this paper, we examine the system to demonstrate how the logical model informs the physical model. We also show how the physical model makes use of the logical model for behavioral logic as well as interactions and demonstrates implementation coverage of the elements defined in the logical model.

The Search and Rescue (SAR) example is taken from the example model document in the UAF specification (OMG, 2021). The purpose of that model was to fully illustrate the various UAF views as a means of showing an example implementation of the UAF specification. While that document sought to inform its readers of the complete UAF specification, this paper concentrates on how the logical and physical models support one another. It also elaborates on some of the elements in that model further than what was done in the example document.

The Electric Quarry

The first example deals with a specific system of systems model. It deals with a quarry where the quarry will be handled by electric machines and where transportation of materiel will be performed by autonomous machines. An overall depiction of the quarry and its basic constituents can be found in Figure 8.

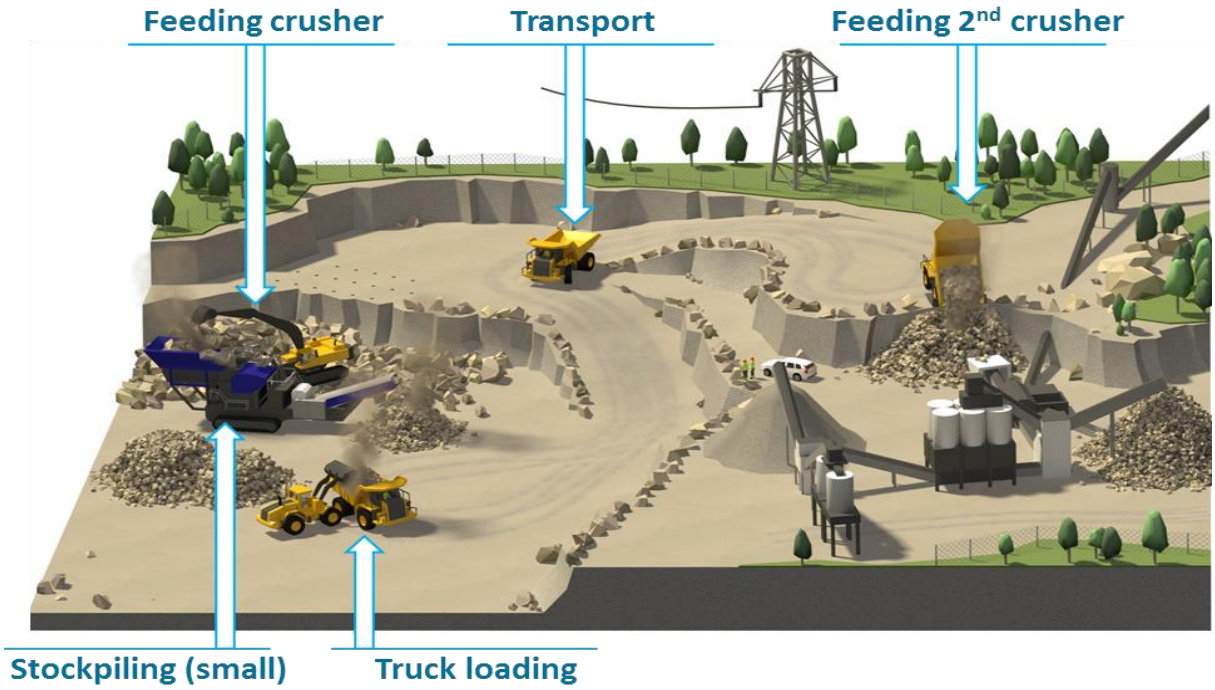

Figure 8. Rock Material Quarry Depiction.

The overall picture above is missing one specific item, namely an electric charger. The reason for performing a model of this system of systems is the need to ensure that the site productivity is maximized, i.e., the amount of materiel processed per hour. To ensure this, it is necessary to turn a system of system into a system. There are several constraints associated with the elements involved.

- The crusher is a stationary electrical device and produces a set amount of materiel to be transported per hour.

- The transporter is electrical as well as autonomous and needs to be charged when the battery is getting low. It can only transport a fraction of the materiel that the crusher creates which implies that a fleet of transporters is required that need to be controlled.
- Loading of the transporter can occur in more than one way: via a loader from a pile of materiel deposited by the crusher or directly from the crusher.
- Loading and unloading takes time and productivity is affected by transporter queuing.

A logical model of the same kind of site can be seen in Figure 9. To determine the needed functionality for managing a quarry many boundaries and known resources are included in the model. As can be seen the model contains several elements that deal with boundary issues:

Known resources: Discard Materiel, Road, Distributed Materiel, Crushed Materiel, External Electricity Grid, Facility Storage, Weather.

All the above has implications for a logical model that aims to ensure that the quarry site can be controlled, and that production can be maximized. This means that from a logical perspective all the elements have behavior, including piles of materiel or roads and crossings. The piles of materiel can report if they are becoming empty or overflowing, the roads are aware of the transporters or loaders that drive on them. This is an abstraction of the realizations that are easily implemented by sensors that monitor the road or the status of piles of materiel. Avoiding these implementation issues here allows the modeler to concentrate on the logic alone.

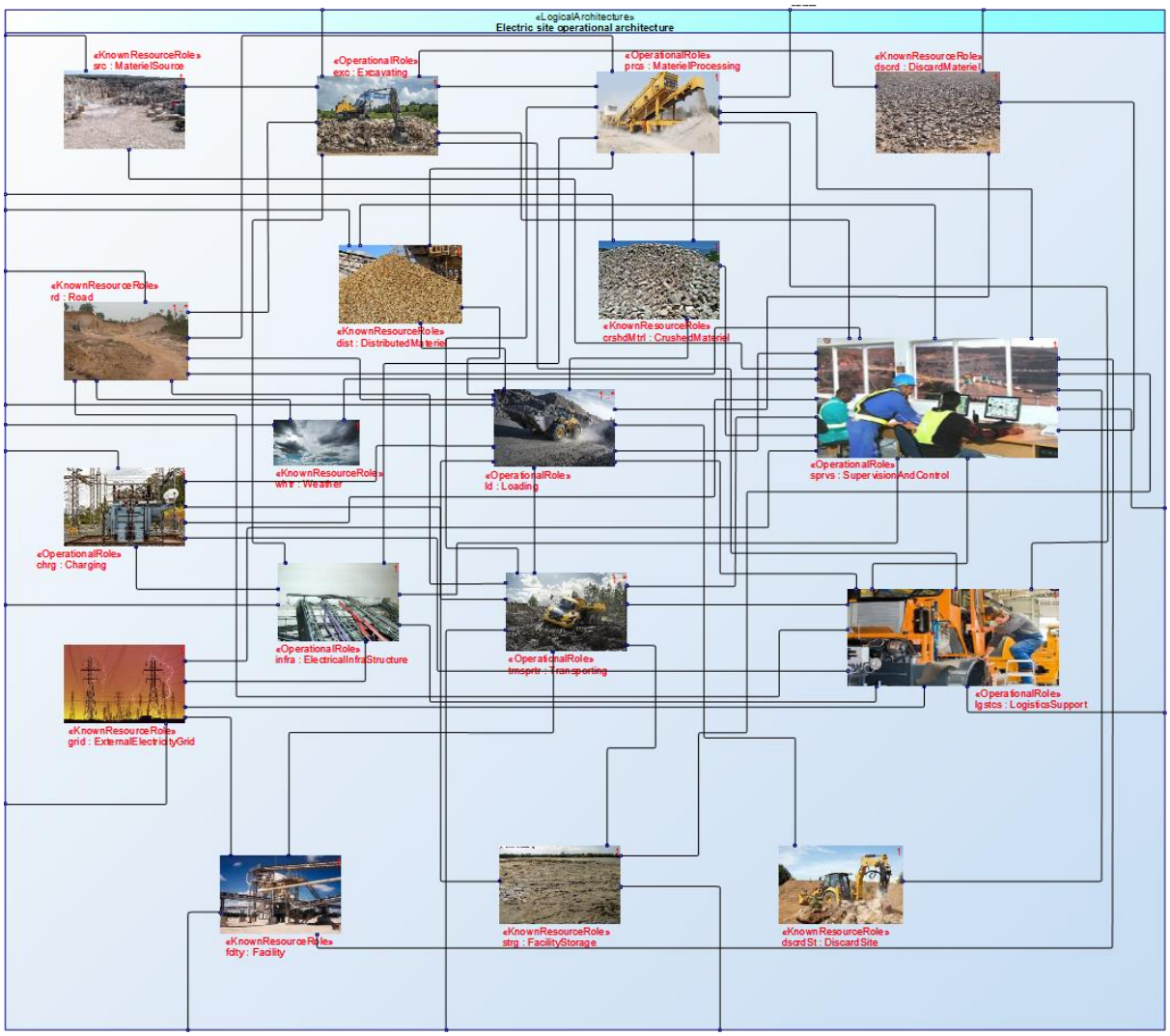

Figure 9. Logical Model of the Quarry.

It is possible to create specifications of the detailed behavior of the different elements in the form of state machines. An example of this is shown in Figure 10 and is concerned with the transporter element in the logical model. The state machine takes a life cycle approach and deals with all the interactions required by the transporter. This takes a birds-eye view of the transporter and focuses on the logic required to perform its functionality within the quarry. It takes several constraints into account such as power consumption while running, time to load, as well as unload, etc.

As can be seen it becomes quite complex when the logic for operations within the quarry for the transporter needs to be defined. The set of data required to define the logical operation also becomes quite large as attributes are added to control state-based behavior. The figure must be enlarged substantially to see what is happening within and in between states. The actions performed within and between states are defined by signals being received, data changes, or periodic time intervals (all defined prior to the "/" sign). As each of these conditions materialize, actions are taken in the form of activities that are shown in blue. They contain logic that causes data concerning the transporter to change as it executes its behavior.

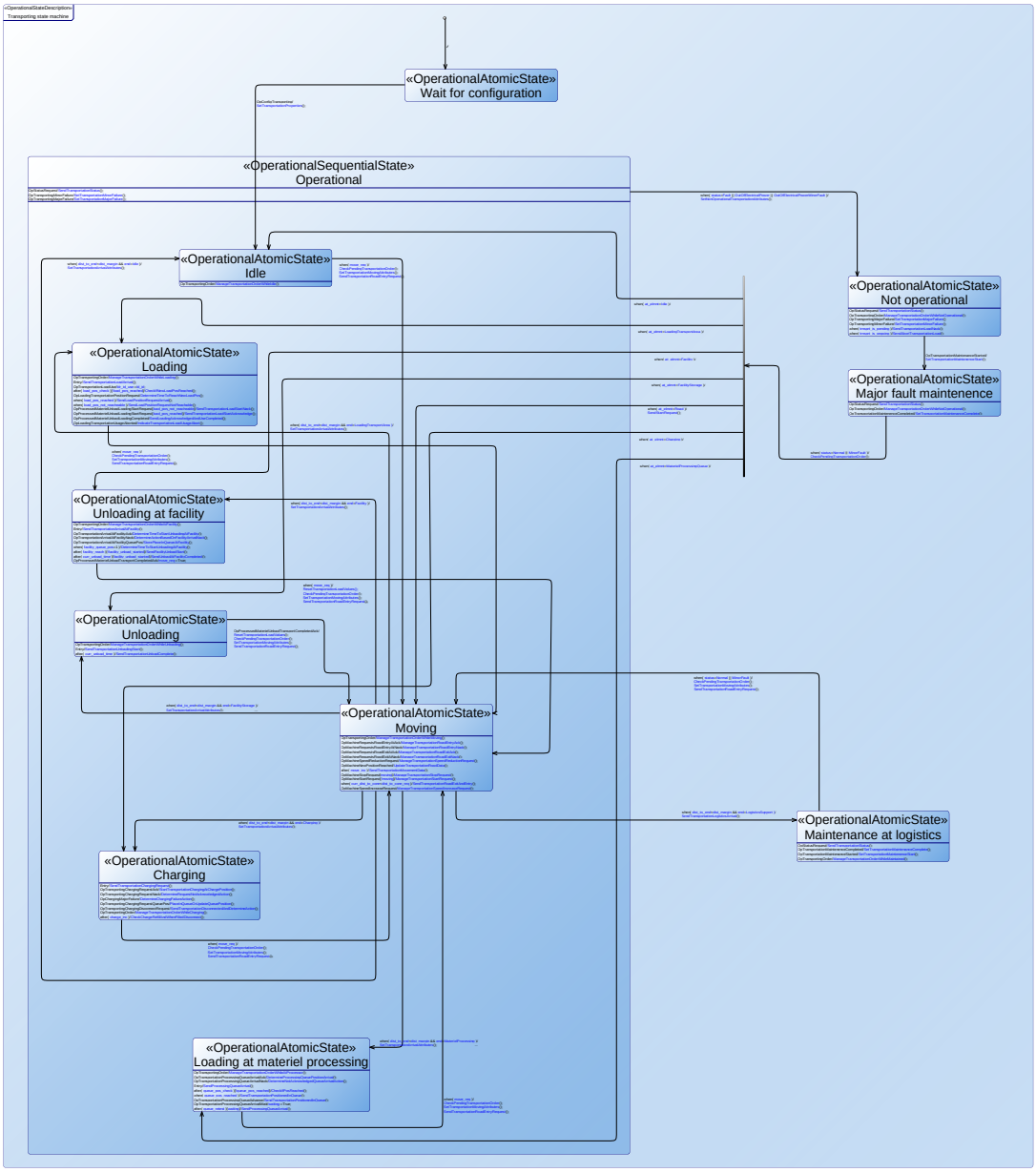

Figure 10. Logical State Machine for the Transporter

In Figure 11 the detailed definition of the data defined for the transporter can be seen. This shows:

- The interfaces that the transporter has.
- Value properties defined for the transporter.
- References to other model elements
- Operations defined for the transporter and finally
- The state machine defined for the transporter.

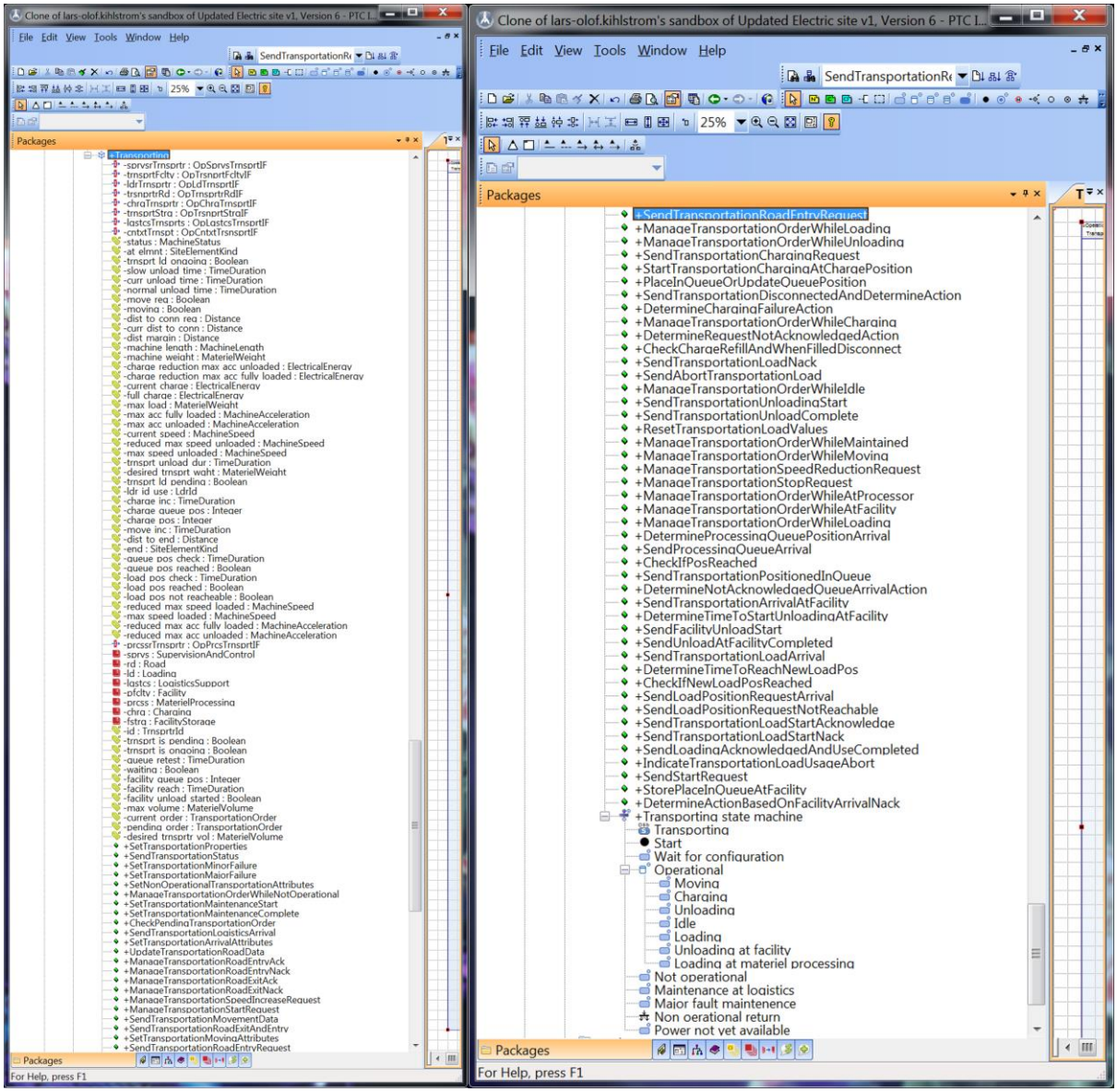

Figure 11. Detailed Model Elements for the transporter.

In the logical model the road, an obvious known resource, is also described by a state machine. This state machine is shown in Figure 12. It can determine if there are transporters travelling on the road as well as where they are. It can also communicate with the transporters themselves to manage them while they are on the road. Logically this is not a problem. It allows the model to focus on the exact handling of transporters on the road without have the issue blurred by the possible implementation. There are several ways in which this can be implemented. The following are examples of ways to realize the functionality required.

- Road sensors and edge computing associated with each road and crossing.
- Centralized control from the quarry controller based on GPS monitoring of transporters and loaders.

- Visually via a monitoring tower, which may be limited if the tower is quite large.

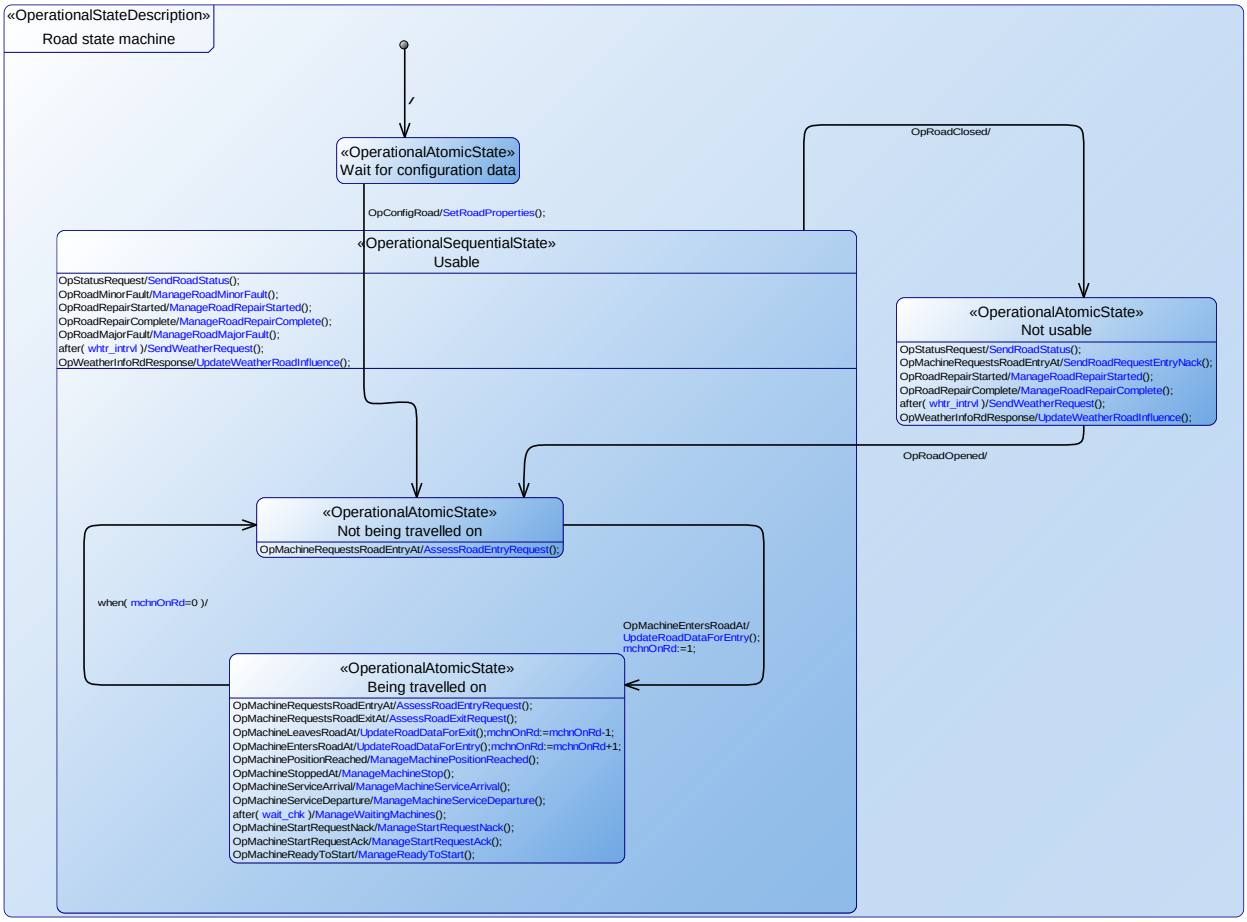

Figure 12. Logical State Machine for the Road.

The physical architecture model for the quarry at the highest level is shown in Figure 13. This looks a lot like the logical model with some differences such as the communication system. This was disregarded in the logical model which concentrated on required exchanges and behavior, and simply assumed communication was possible.

It becomes very similar simply due to the need to take the same kinds of constraints as well as known resources into account that the logical architecture was required to do. There are however additional implementation-based constraints such as the fact that while the logical architecture could focus on the needs for exchange of information in between the logical entities, the physical architecture needs to take the implementation of this communication into account. From an implementation perspective, the logical materiel processing part was divided into two parts, the crusher as well as the mobile conveyor.

Naturally, the known resources here are not implemented as such but can be subdivide3d into different parts that can be implemented and provide the logical functionality needed. The piles of materiel can be monitored by monitoring equipment to ensure that they do not overflow. The weather can be monitored such that any special handling of inclement weather can be dealt with and as indicated the roads can be monitored by special equipment.

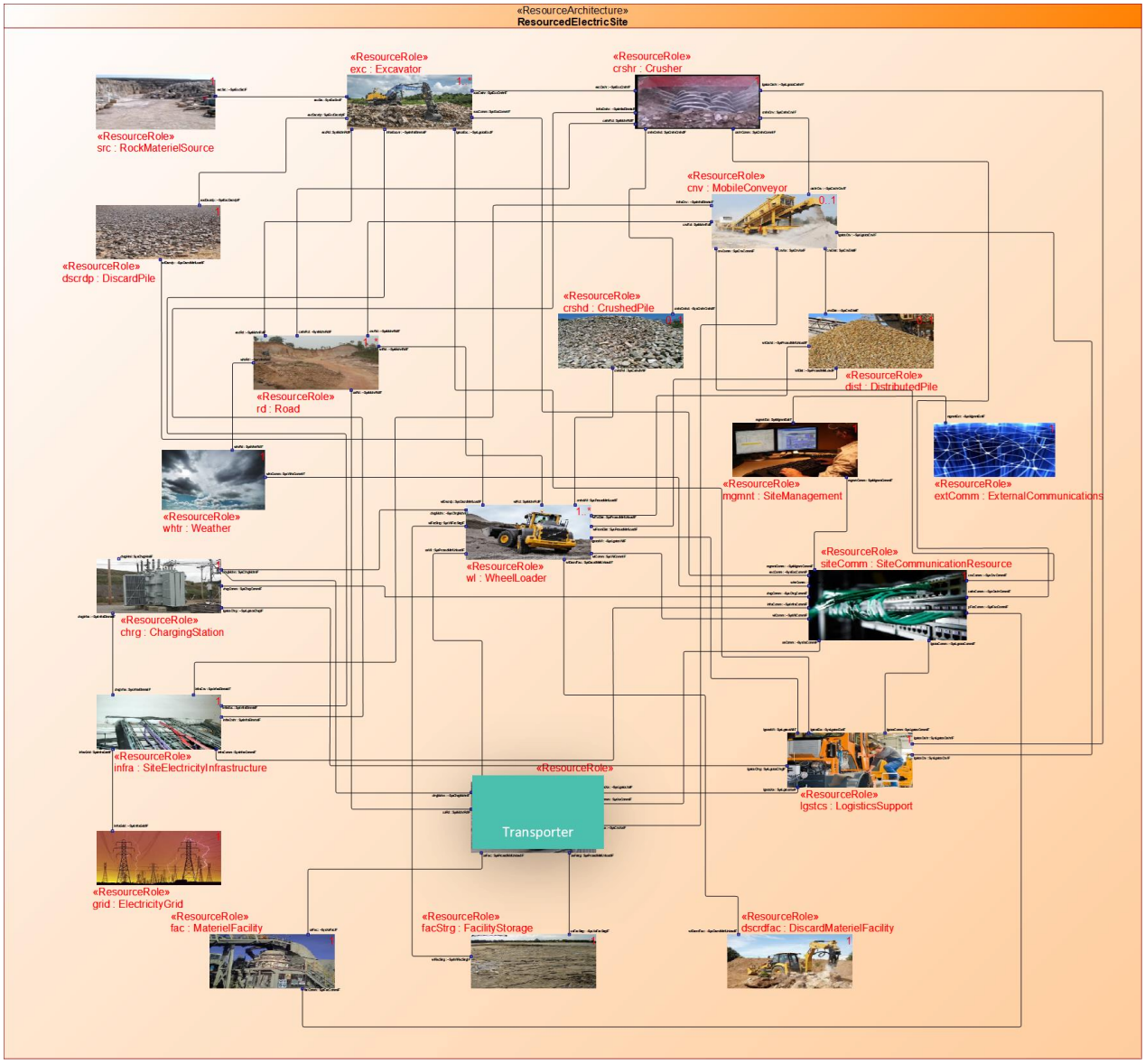

Figure 13. Physical Architecture of the Quarry.

In Figure 14, a physical architecture breakdown can be seen for the transporter. It has been broken down into the following parts:

- Transporter mchn: The machinery parts of the transporter, hardware, and software.
- Auto: The transporters internal autonomous control equipment, hardware, and software.
- Sensors: The sensor parts of the transporter, hardware, and software.
- Comms: The communications parts of the transporter, hardware, and software.
- HMI: The human machine interface, i.e., controls that allow a human to interact directly with the transporter. The transporter does not have a driving compartment as such, an interface with manual controls as well as displays may well exist.
- Qpilot: The part of the transporter devoted to managing the interactions with other parts of the quarry, i.e., the hardware and software required to work with other elements within the quarry.

A state machine has been described for the Qpilot part is shown in Figure 15.

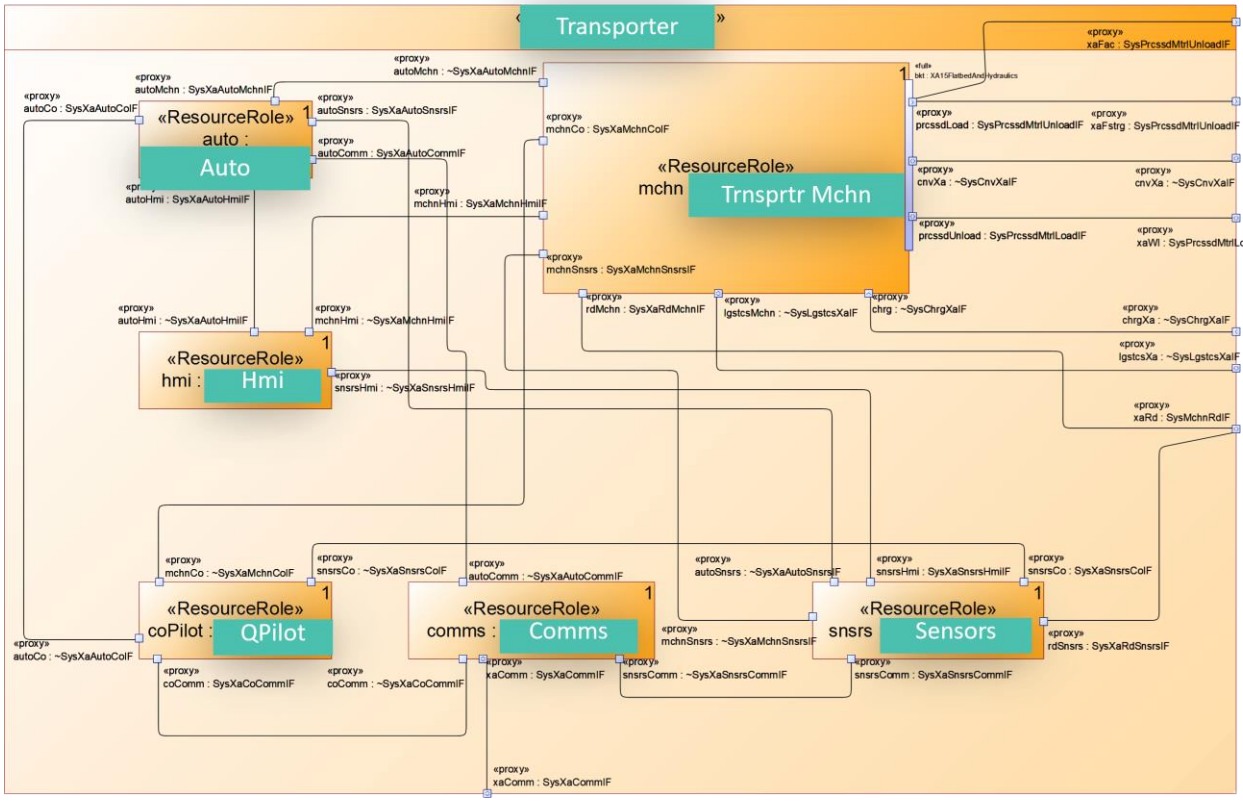

Figure 14. Transporter Physical Resource Architecture.

Comparing the logical state machine to the physical state machine shows extensive similarities between the two. This means that if the logical state machine had been validated by means of simulation, a considerable work reduction will take place in the actual software implementation of the elements dealing with creating the software required to run the transporter within the quarry. The above connection exists for all the elements that have a counterpart within the logical model.

The match is not 100% since the Qpilot state machine must take inputs from the other parts of the transporter into account, i.e., there are additional pieces of information that needs to be dealt with that the logical state machine did not have to deal with. Some of the logical interactions described in the logical state machine will appear within the Qpilot state machine mediated by the external elements within the transporter.

This logical state machines were created by looking at each of the elements in the logical model from outside, taking constraints and known resources into account. These were then transformed into the functionality required and defined as state machines within the physical elements. This transformation can be used to ensure functional compliance in a real implementation. It will also reduce the amount of implementation work as the state machine clearly defines the required behavior far better than a text description ever could.

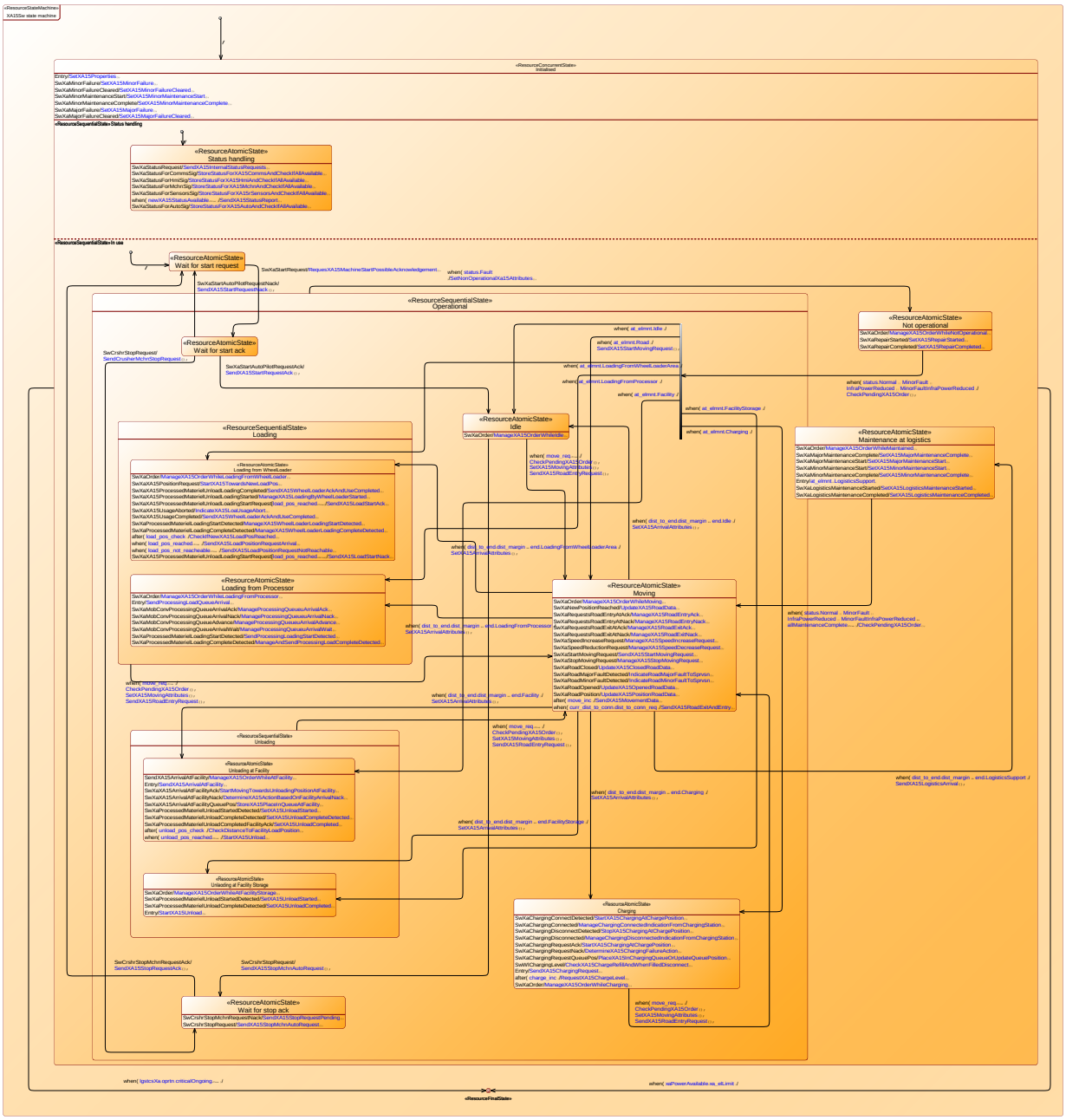

Figure 15. Transporter Qpilot State Machine.

Search and Rescue (SAR)

Further examples can be found in the example model used in the UAF version 1.2 standard. It deals with Maritime SAR (MSAR) and is an adapted version to demonstrate logical/physical mapping. Search and rescue in a maritime setting has been around for a long time usually in a very simplified form as an example of architecture modeling. The adapted logical model of the MSAR example can be seen in Figure 16.

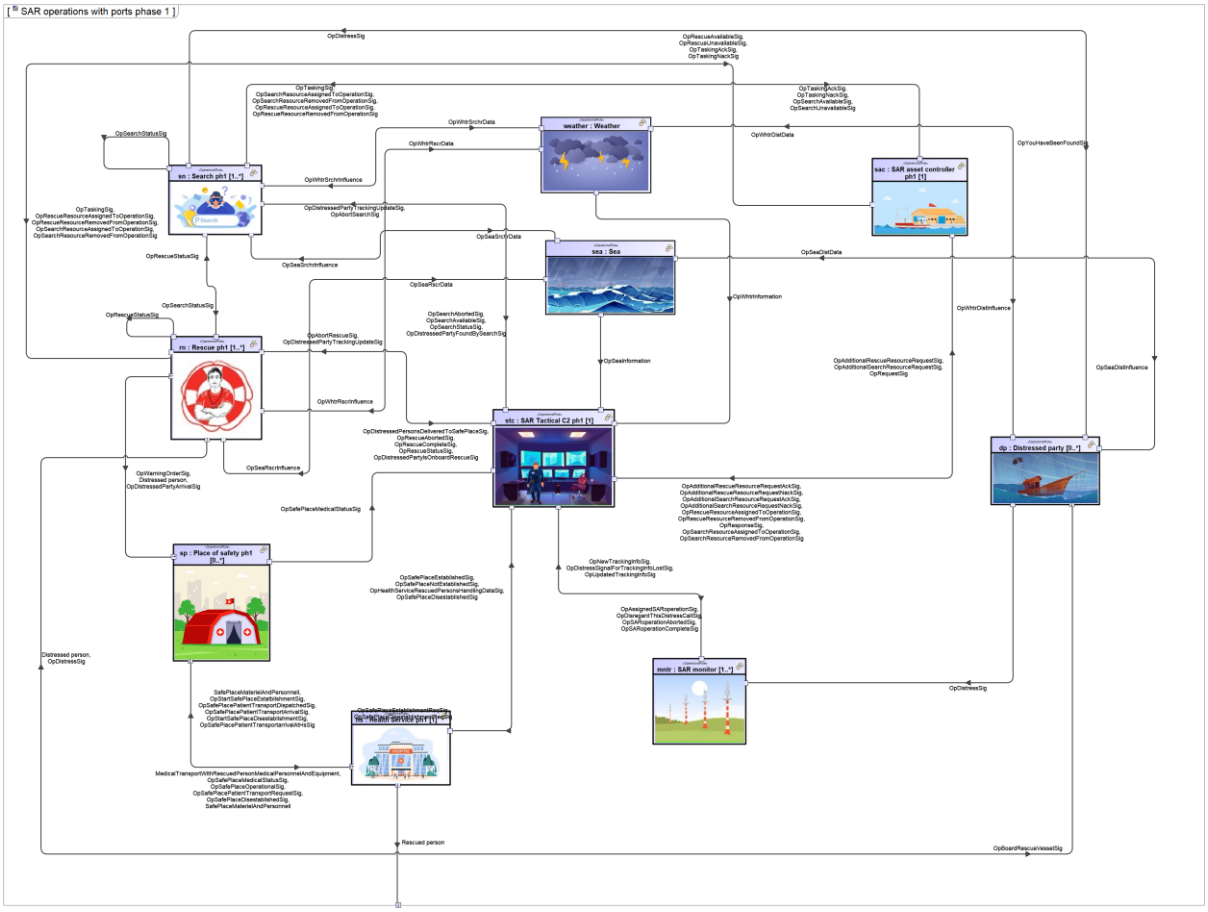

Figure 16. Maritime Search and Rescue (MSAR) Logical Model.

The additions to the original simple model are due to the need to deal with constraints as well as logical known resources. They can be summarized as follows:

- The influence of the sea as well as the weather needs to be considered for search, rescue, and the distressed element.
- The interactions required with the known resource National Health Service (NHS) needs to be considered.
- The handling of more than one MSAR operation at one time has been added.
- Depending on the kind of distress, several instances of the search and rescue elements operating under their own constraints such as range of operation, number of persons that can be handled etc., needs to be considered.
- As the situation changes within an operation, more search and rescue assets may need to be either added or withdrawn from an operation.
- Search or rescue asset failures during an operation need to be considered.
- A tactical Command and Control (C2) node is needed to manage operations.
- An asset resource base is needed from which search or rescue assets can be requested.

The logical state machine that deals with search is shown in Figure 17. The state machine has been created such that all the above constraints have been dealt with for the logical search asset. Some of the actions required have been placed in concurrent regions since they need to be dealt with irrespective of where in the search state the search element happens to be in. The semantics of a concurrent states within a region of a state mean that they are executed at the same time, or concurrently. These are status handling, search, and resource handling regions.

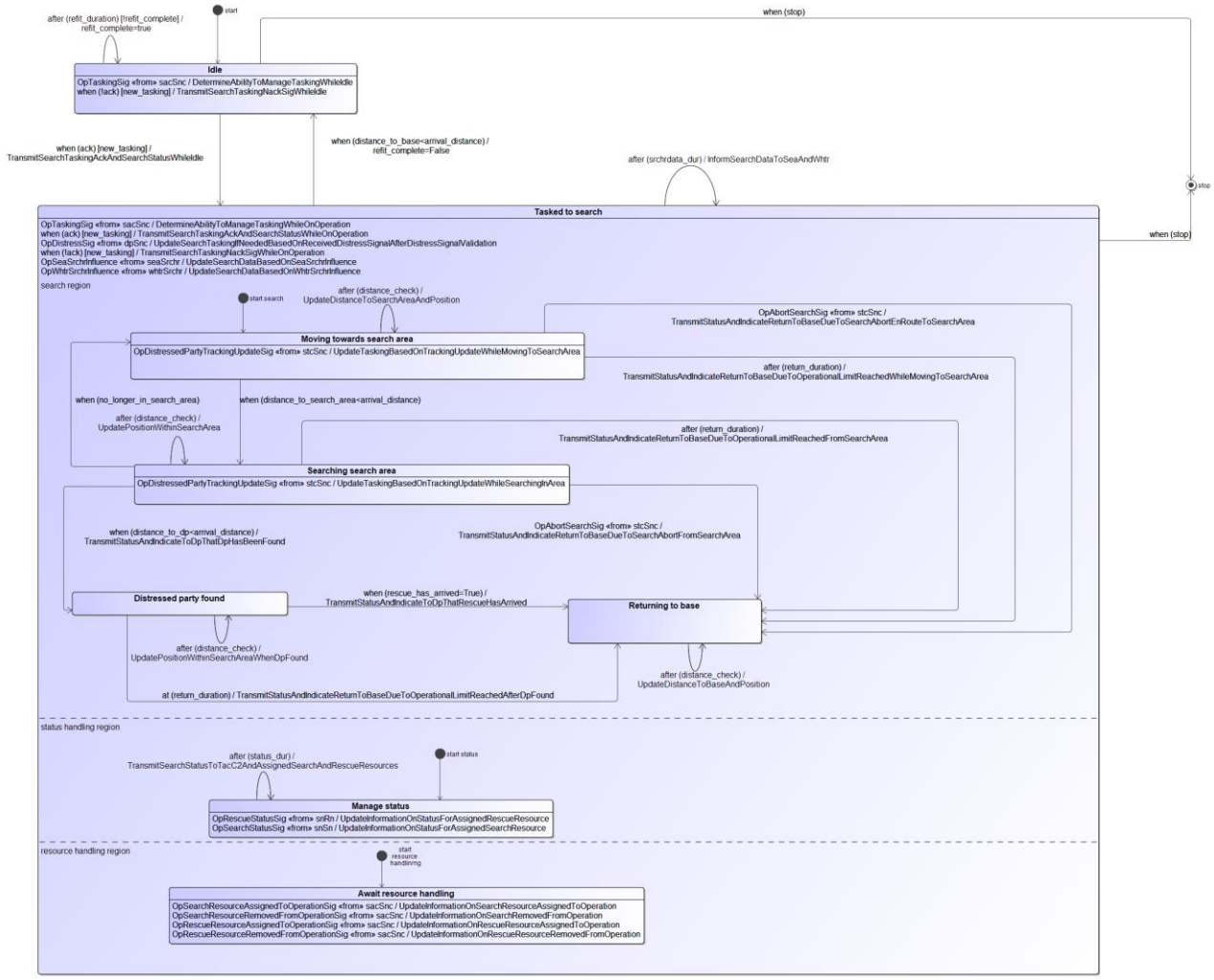

Figure 17. Logical State Machine for the Search element in MSAR.

Within a physical resource architecture there are several kinds of elements that can act as realization of the search and rescue elements in the logical model. Figure 18 shows candidate implementations of the search and rescue elements. The SAR helicopter and the SAR ship implement both the Search and Rescue logical elements. The SAR search aircraft as well as the SAR search drone only implement the search element as they are not equipped for rescue.

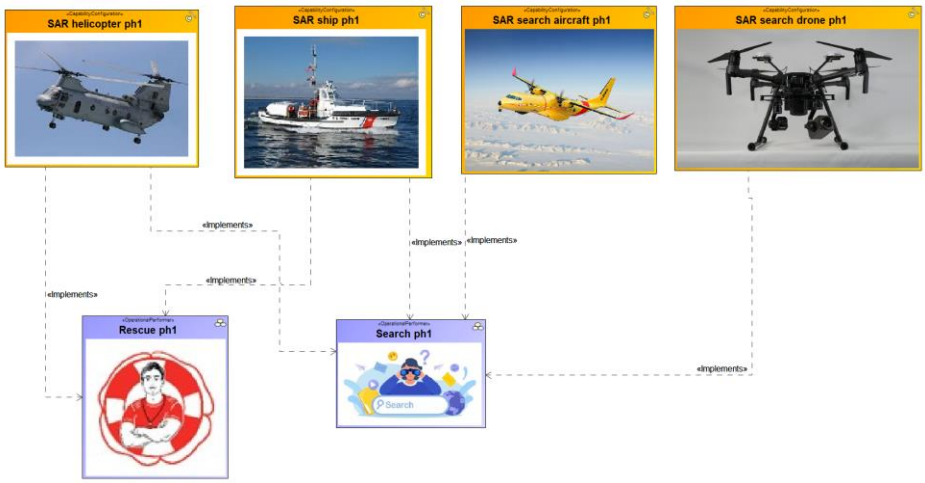

Figure 18. Physical Resource Implementation of Logical MSAR Elements.

The same kind of relationship as described for the Transporter in the quarry exists for the different implementations of the search and rescue logical elements in the maritime SAR system. As previously, each implementation can be broken down into a lot of different parts, containing both hardware and software. The part that specifically deals with the search and rescue operation context can reuse the logic described in the logical architecture to a high degree. The reuse id not 100% but sufficiently large to have a distinct impact on the realization work effort.

A complete physical architecture for an MSAR operation is shown in Figure 19.

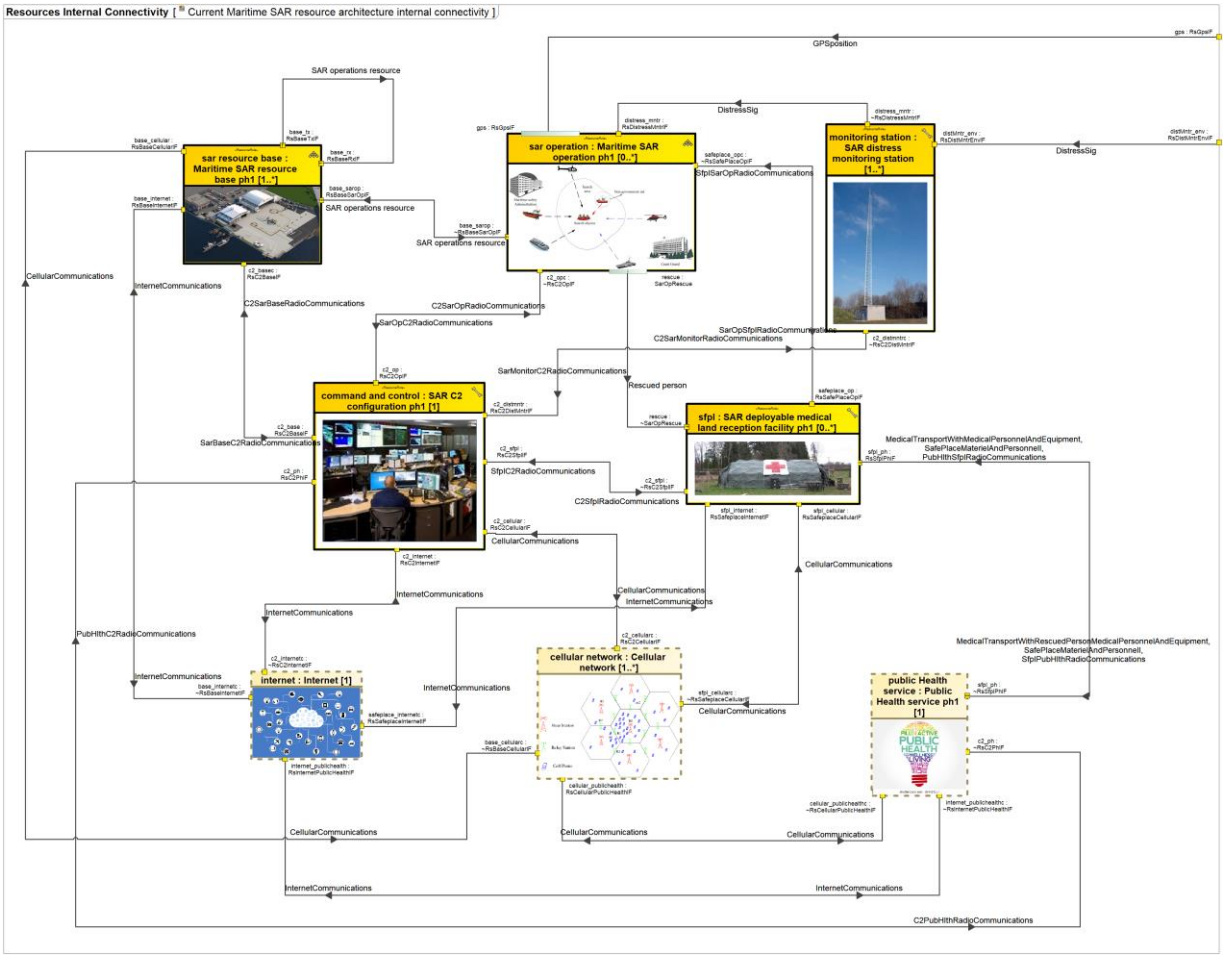

Figure 19. Complete MSAR Physical Architecture.

Some of the external and internal context is shown in the above figure such as the asset base from which search and rescue assets can be allocated, the command-and-control center, monitoring stations as well as deployable medical reception facilities. It also shows possible external communications such as cellular networks and the Internet made use of by the MSAR architecture and finally the external public health service.

The MSAR operations part of the complete MSAR physical architecture can be further expanded as shown in Figure 20. This shows the physical systems associated with the MSAR operation (aircraft, drone, helicopter, ship), possible forms of distressed maritime vessels and persons, sea and weather that needs to be taken into account and also the radio communications environment through which all of the communications in between the MSAR assets needs to pass.

In the same fashion as for the quarry the drone architecture consists of different parts as shown in Figure 21:

- Drone radio communication system
- Drone tactical data link
- Drone aviation system
- Drone GPS
- Loudspeaker, floodlight, radar, and visual/ IR detection systems
- Drone control system: this being the system tasked with the management of the drone within the maritime search and rescue context.

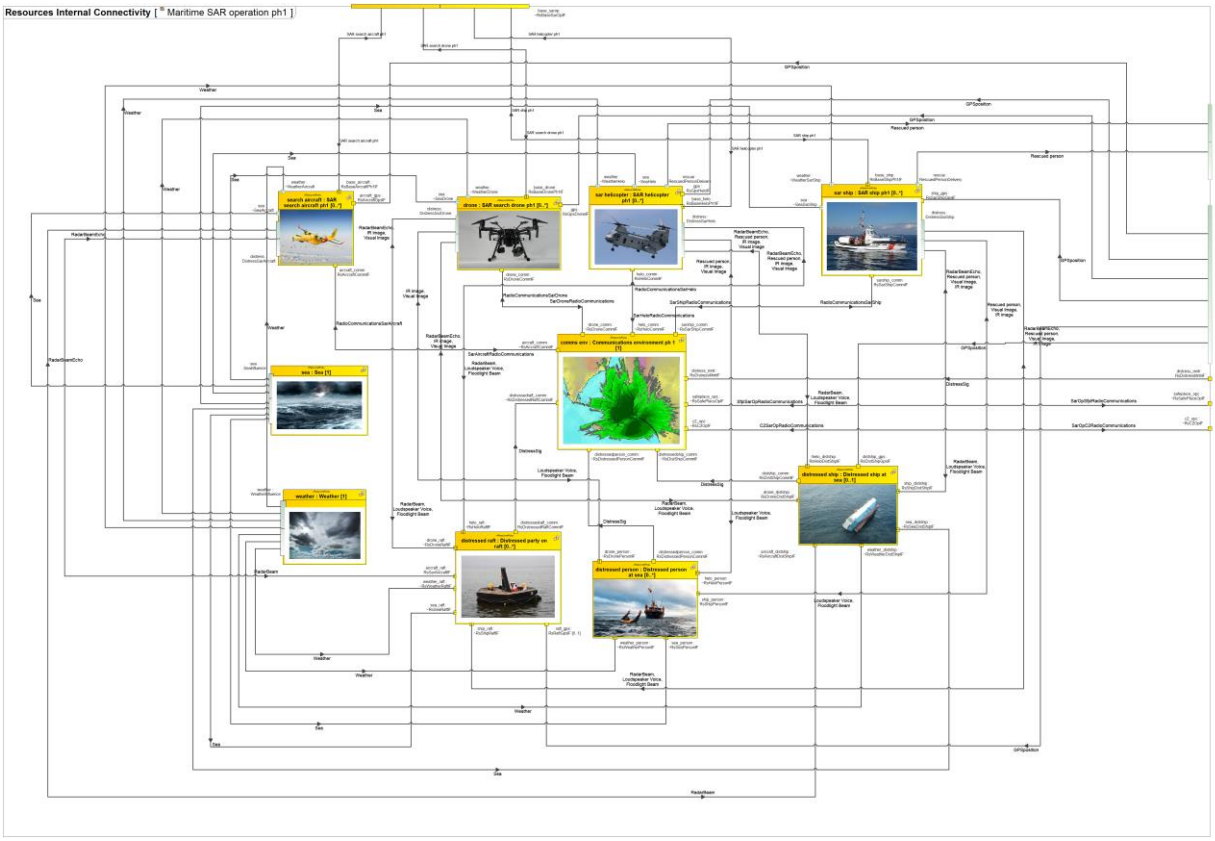

Figure 20. The Maritime SAR Operation.

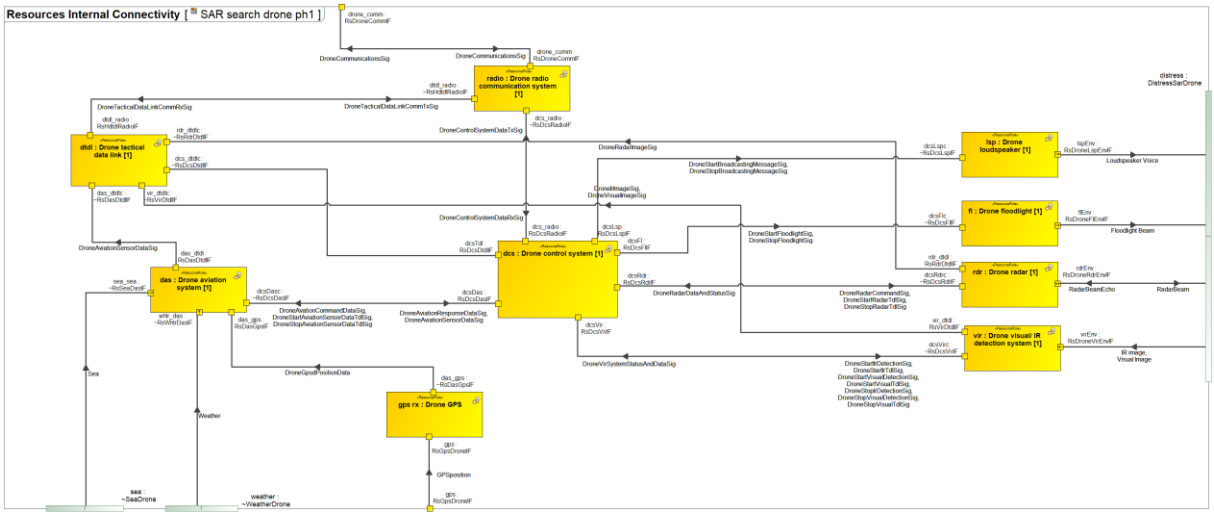

Figure 21. Resource Architecture Diagram for SAR Search Drone.

It is then possible to expand the Drone control system into a set of adaptors to provide interfaces to the external systems and the actual control software for the drone for MSAR as shown in Figure 22.

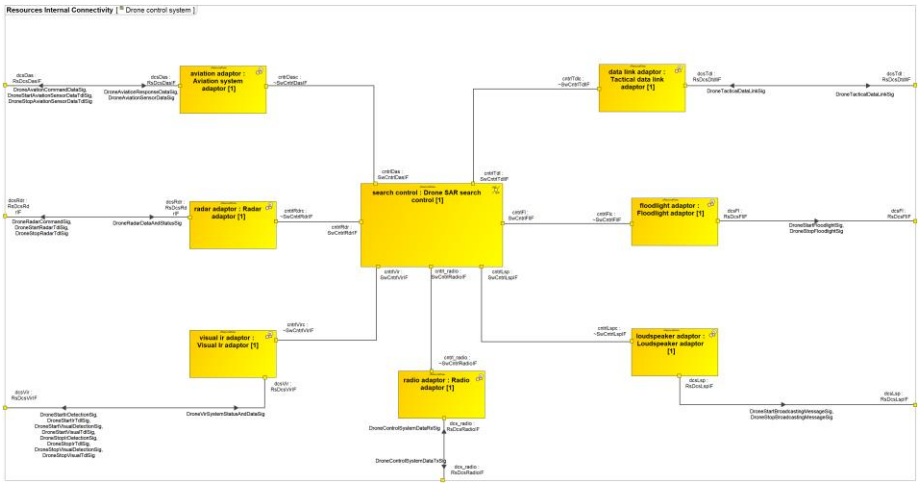

Figure 22. SAR Drone Control System Components.

Finally, the state machine for the Drone SAR search control can be defined as shown in Figure 23.

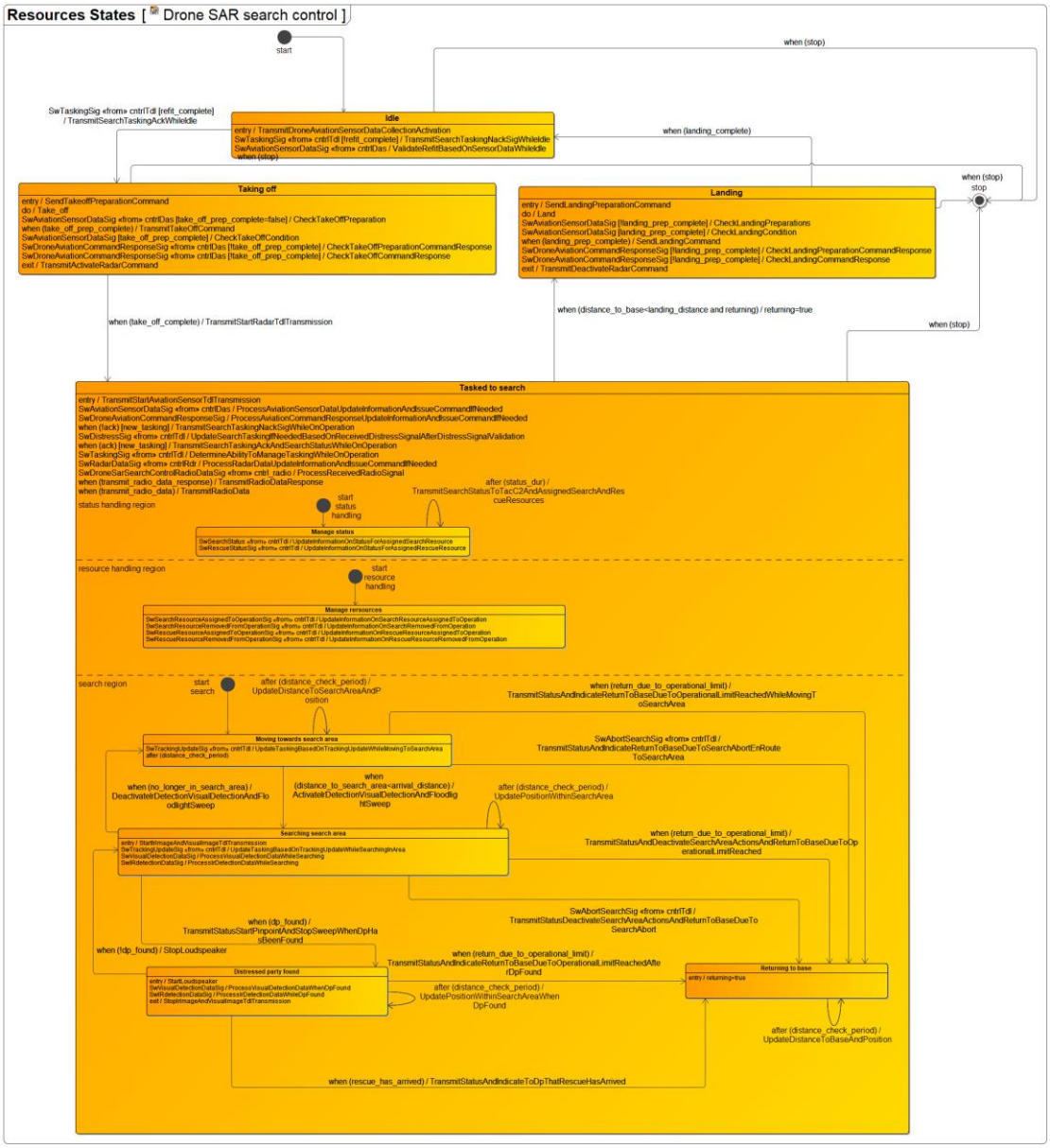

Figure 23. Drone SAR Search Control State Machine.

When comparing the logical search element state machine with the Drone SAR search control state machine the similarities are obvious, especially within the region named search. This region is expanded further in Figure 24. The similarities are not 100% but are large enough to warrant a detailed handling of the logical model and ensure that the logical model is validated by means of simulations of various possible scenarios.

The differences between the state machines result from the logical element abstractions such as the information interchange between logical Search and Logical Weather and Sea. From a logical point of view this allows testing of different scenarios, but the influence will be dependent on data from the logical elements (search, rescue as well as distressed party) and this is why this interaction is included in the logical model. The influence that weather has on the physical implementation of search in the form of the drone end up as data from the drone aviation system. It may seem that since the drone is airborne, the impact from sea would be less but the ability of the drone to detect a distressed party may well be impacted by the sea and could require handling of both radar, infrared detectors as well as the floodlight.

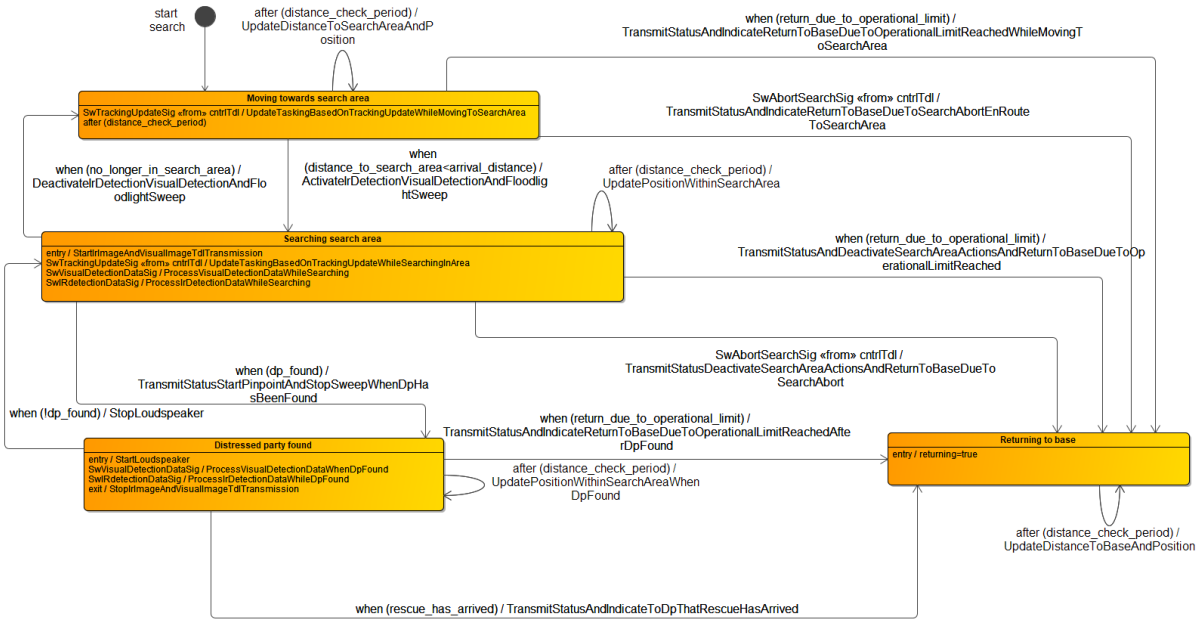

Figure 24. Search Region of Drone MSAR Control Software.

Conclusion

The examples shown here attempts to illustrate that the relationship between an operational/logical model and a physical realization model is quite close. It is quite common however to view the logical model as an initial analysis that is then left once the realization work begins. Based on the examples given in this paper, an attempt has been made to illustrate that there is a great benefit in treating the logical architecture as being as important as the physical architecture and that it should be maintained during the lifetime of the system of interest to the same extent as the physical architecture.

A logical architecture that takes proper account of real-world constraints as well as known resources will make it possible to speed up the development of the physical realization. If the logical architecture is used as the basis for simulation, logical problems can be identified and fixed prior to the detailed implementation. The logical architecture will also directly influence the detailed development within the physical realization.

The logical architecture also serves as a way of quickly getting to grips with requirement changes for the system of interest as well as understanding the impact of environmental context changes.

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.

Farcas E., Farcas C., & Krüger I., 2014, Economics-Driven Software Architecture, Chapter Chapter 12 - Successful CyberInfrastructures for E-Health, Published by Morgan Kaufmann

Friedenthal, S., Moore, A., Steiner, R. Practical Guide to SysML: The Systems Modeling Language Second Edition, Morgan Kaufman, Oct 31, 2011

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., 2022, You Can't Touch This! - Logical Architectures in MBSE and the UAF, presented at the 2022 INCOSE International Symposium, Detroit, Michigan, USA.

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

INCOSE 2021, Systems Engineering Body of Knowledge, SEEBOK

ISO 15288 2023, Systems and Software Engineering — System Life Cycle Processes, International Standards Organization (ISO), ISO/IEC/IEEE FDIS 15288-2023.

ISO 42010 2022, Software, Systems and Enterprise — Architecture Description, International Standards Organization (ISO), ISO/IEC/IEEE 42010-2022.

ISO 42020 2019, Software, Systems and Enterprise — Architecture Processes, International Standards Organization (ISO), ISO/IEC/IEEE 42020-2019.

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

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

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

OMG 2017, *Unified Modeling Language, Version 2.5.1*, Object Management Group, https://www.omg.org/spec/UML/2.5.1/About-UML.

——— 2019, *Systems Modeling Language, Version 1.6*, Object Management Group, https://www.omg.org/spec/SysML/About-SysML/.

——— 2022a, *Unified Architecture Framework, Version 1.2*, Object Management Group, https://www.omg.org/spec/UAF/About-UAF/.

——— 2022b, *Unified Architecture Framework Modeling Language, Version 1.2*, Object Management Group.

——— 2022c, *Enterprise Architecture Guide for the Unified Architecture Framework (Informative), Version 1.2*, Object Management Group.

——— 2022d, *Unified Architecture Framework Sample Problem (Informative), Version 1.2*, Object Management Group.

Sjöberg P., Kihlström L., Hause M., 2017, An Industrial Example of Using Enterprise Architecture to Speed Up Systems Development, INCOSE International Symposium, Adelaide, Australia

Biography

Lars-Olof Kihlström. Lars-Olof Kihlström is a principal consultant at CAG Syntell 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 since the middle of the 1980's such as telecommunications, automotive, defense 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 is an SSI Principal and MBSE Technical Specialist, a former PTC Fellow, a co-chair of the UAF group and a member of the OMG SysML specification team. He has been developing multi-national complex systems for over 45 years as a systems and software engineer. He started out working in the power systems industry and has been involved in military command and control systems, SCADA, distributed control, office automation and many other areas of technical and real-time systems. His roles have varied from project manager to developer. He has written over 100 technical papers on architectural modeling, project management, systems engineering, model-based engineering, human factors, virtual team management, product line engineering, systems of systems, SysML and Architectural Frameworks such as UAF, DoDAF and MODAF. He is a proud recipient of the INCOSE MBSE Propeller Hat Award.

.