An approach to SysML

Contents

[Release History 2](#_Toc15115193)

[Definition of Terms 2](#_Toc15115194)

[Assumed knowledge 3](#_Toc15115195)

[Summary of Article 3](#_Toc15115196)

[Intoduction 3](#_Toc15115197)

[What is a Model 4](#_Toc15115198)

[Review of Requirement and Solution Terms 8](#_Toc15115199)

[Divide and Conquer Strategies 9](#_Toc15115200)

[Domains 10](#_Toc15115201)

[How to find the service domains of a system 11](#_Toc15115202)

[The Architectural Domain 12](#_Toc15115203)

[Domain Properties 12](#_Toc15115204)

[Domain Relationships 13](#_Toc15115205)

[SysML concept of Allocation 14](#_Toc15115206)

[Synchronisation Specification 15](#_Toc15115207)

[The Control System Model 16](#_Toc15115208)

[Entity Relation Model 18](#_Toc15115209)

[The V Model 19](#_Toc15115210)

[The Interface Model 20](#_Toc15115211)

[The Continuous Process Model 21](#_Toc15115212)

[Further Entity Relation Diagram Examples 24](#_Toc15115213)

[Summary 29](#_Toc15115214)

[Feedback 30](#_Toc15115215)

[References 30](#_Toc15115216)

[Notes 30](#_Toc15115217)

[Issues 30](#_Toc15115218)

# Release History

4/7/2017 v1.0

7/1/2018 v2.0

6/3/2018 v2.1

13/5/2018 v2.3

25/7/2018 v2.4

27/7/2019 v2.5

3/8/2019 v2.6

3/3/2020 v2.7

# Definition of Terms

Abstract – An object or behaviour derived from a set of real objects

Abstraction – A process that produces an abstract object from a set of physical objects

Abstract Object– Represents a requirement and a set of solutions in a SysML model

AI – artificial intelligence

Allocate – The sharing out of a resource or assignment of behaviours to objects

Application – The primary task of the System which benefits the user.

Application requirements model – The requirements of the application in model form

BDD – Block Definition Diagram

Behaviour – Describes changes in time

CASE – Computer Aided Software Engineering

COTS – Commercial off the shelf

Derived – Something that is created by analysis

Design – The act of creating a solution satisfying a set of requirements

Domain – Extent of the subject matter being modelled.

Domain – A world inhabited by a distinct set of objects which characterises the domain.

[DOORS - Dynamic Object-Oriented Requirements System](http://www.rcm2.co.uk/products/DOORS.htm)

ERD – Entity Relation Diagram. This is the physical object diagram

EPR – Engine pressure ratio

FADEC A full authority digital engine (or electronics) control

HTTP – Hypertext Transfer Protocol

IBD – Internal Block Diagram

Liveware – that part of the sytem where operations and functions are carried out by people

Logical – Descriptions of the physical world in terms of abstract concepts

MBSE – model based systems engineering

MMI – Man Machine Interface

N1 – rotational speed low pressure compressor

N2 – rotational speed high pressure compressor

OO – Object Orientated. Logical model of the real world

OSI – Open Systems Interconnection

Peopleware – see Liveware

Physical – Descriptions of the physical world in terms of existing designed components

Protocol Stack – implements a standard communication standard in software

Requirement – Specification that a solution must satisfy

ROI – Return on Investment

RUP – Rational Unified Process

Service – Supporting mechanism of the primary purpose of a system

SME – Subject Matter Expert

SOA – Service Orientated Architecture

Solution – Something that solves a problem specified by requirements

Solution set – A number of different solutions which solve the same problem

Stakeholder – A person with a controlling interest

Structure – Physical objects which are often interconnected both physically and electrically

Sub- System – Arbitrary division of a System

Subject Matter – Study of a particular topic.

SysML - Systems modelling language

System – Group of coordinated functions that are complete and have a single purpose

80% 20% rule – The cost of producing the first 80% is the same as producing the last 20%

# Assumed knowledge

1. SysML
2. Design of real time Systems
3. Control system design

# Summary of Article

This article is about the design of real time systems using SysML. Its describes an approach to modelling which gives the reader an understanding of the concepts on which SysML is based, The guide shows a pragmatic way to model using a conquer and divide strategy based on domains.

# Introduction

SysML is a language specifically designed for[[1]](#footnote-1) systems which exhibit large behavioural characteristics. This is the opposite of an IT system which typically handles only retrieval and storing of data. The language has been developed from UML and covers the modelling of system components comprising hardware, software, firmware and people behaviours (Live ware).[[2]](#footnote-2) Typical examples include lift systems, traffic light systems, car parking systems, jet engine controllers, robotic systems, air traffic control systems, factory processes, power steering etc. Imbedded real time systems having large centralised control elements implemented in software can benefit from the use of SysML.The SysML model is very good at capturing behavioural aspects of the sytem as requirements placed on abstracted objects. The SysML can be used to address other aspects of the system design.[[3]](#footnote-3)

SysML unfortunately can suffer the consequences of inappropriate use arising from miss-understanding of its purpose and place in the overall system lifecycle. For example consider a hazardous environment where objects are moved about using a robotic system. The requirements may specify thousands of movement increments needed by the robot to pick up the payload, move it and place it at the destination. The team starts perhaps by modelling those steps only to find that the model is identical to data in the incremental steps document. The team soon finds that those steps are not fixed but can have a large number of alternatives. By this time everyone is wondering what is actually being modelled, how to make the mandated allocation diagrams and why the logical model appears to be so concrete. More over this approach is quickly leading to duplication of effort with other teams doing the same thing but using traditional methods. There is a rush to find a solution and customer pressure. The problem in this instance is that information in the database is fixed and is the final solution specification. This “bad smell occurs when the model has no further elaboration possibilities for optimising the solution..

SysML is often used when other tools, which are specifically designed for final solution specification, are more appropriate. The opposite is also true. For example Matlab and Simulink are used for requirements when the use of SysML would have been more appropriate. This wrong choice of tool leads to poor ROI[[4]](#footnote-4) particularly in legacy projects. The extra information produced will be a duplication of existing traditional documentation and will not move the project forward.

The take up SysML could be improved if a way of getting more for our money could be demonstrated. Too often a too shallow understanding of the worth of a model, how and where SysML should be deployed if at all, leads to frustration among project engineers and the abandonment of its use.

This write up examines the role that SysML should play and what it can be used for in a project by re-examining the concepts on which SysML is based.

We start by examing what a model is.

# What is a Model

A model is a representation of a real thing. In this context a SysML model can be made to represent a system in exactly the same way as a model ship represents a real ship. The process of abstraction is used in modelling, Abstraction achieves a reduction in detail which leaves only those things which are essential making the model simpler and therefore easier to understand. The word abstraction means to view a model as a set of requirements rather than a set of solutions. The problem for the modeller in how to represent all requirement details in a single model. The SysML model can be broken down into a series of models describing different aspects of the system. These separate models are found by deploying a divide and conquer strategy which breaks the detail into manageable chunks.[[5]](#footnote-5) Details are discovered by thinking about the solution or design process. For example the details of robots movements described above only come to light after deciding that a possible solution might be to deploy a robot.

The SysML model can be hierarchical. At a particular level the model serves as a description of the solution of the requirements specified in the level above. This solution description is also the requirements of some as yet unspecified solution to be shown in the level below. The bottom level of the model are the lowest level of requirements covered by the SysML model,[[6]](#footnote-6)

The work of requirement analysis may have already taken place and mistakenly the SysML modelling is introduced as an after thought to help with solution details. This is the wrong approach as all levels of the model represent the requirements. It is this fact that makes the modelling difficult since requirements are not something that can be conjured up over night, It takes analysis effort to make a model useful and understandable.

The aim of a model is to make the requirements more unambiguous and less open to interpretation compared to written or spoken words.[[7]](#footnote-7) For example it is difficult to pin down precise behaviour just using story boarding, use cases or agile descriptions. Requirements as written statements are always less precise and open to interpretation than say a SysML activity diagram.

The use of SysML can be seen as an exercise in requirement analysis. That is trying to discover the requirements for a system so that it can be built. It is sometimes thought that requirements need to exist before a model can be constructed, This is incorrect. A SysML model by itself can be a complete description.

It is not usual though to use SysML for requirement specifications which involves business analysis and justification. Most projects will have written requirement statements outside the SysML model. In a typical project many will not be using SysML. A useful exercise, in support of modelling, is to construct sentences from the model and so make a more complete set of written requirement statements.

Sometimes in legacy systems those written requirement statements are good enough for the job in hand and no further requirement analysis is needed. In this case a SysML would be duplication of effort. It is the responsibility of the project manager to way up the cost and assess whether any further requirement analysis will result in a ROI.

The SysML has been designed to bring the best possible outcome for requirement descriptions. That is the user of the SysML model is able to make progress from specifications found in the model. Object blocks[[8]](#footnote-8), sequence diagrams, activity diagrams and state machines are used to help further elucidate behavioural requirements, SysML is also used to represent the requirements for how systems can to be built. This involves allocating behaviour to objects representing the system architecture.

A common miss-conception of the nature of a SysML model is to think of it as dynamic. Use cases and activity diagrams give the impression that the model is showing a sequence of ordered steps through time. Both use cases and activity diagrams only represent the possible system state-event combinations. In order to move the system from one state to another a set actions are also specified. To make sense of the state event matrix the main scenarios are represented in the model by sequence diagrams.[[9]](#footnote-9) The model does not say how objects change state or how the actions are achieved. The model is a statement of the requirements and as such is incapable of expressing deficiencies which might occur in the implementation. A model has perfect and instantaneous behaviour. The system behaviour under fault conditions is generally not modelled because domain analysis, see below, would rule that out.[[10]](#footnote-10) In this sense a model is never wrong but it may express requirements which others do not agree with.[[11]](#footnote-11)

The diagrams below show a model for producing water at 100OC a water boiler. The usage scenario or use cases are used to derive the state event matrix for the water batch shown below.



Figure State Event Matrix for the water batch

Two requirements of the system are shown in the diagram below and are traced to the object water batch.

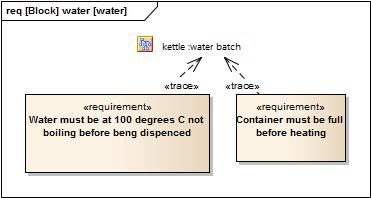


Figure 2 Example requirements traced to the state machine

The sequence diagram shows the objects making up the kettle and its contents. The objects have state and the diagram shows how the states are synchronised. For example when the batch is in the heating state the container is in the full state and the heater is in the on state.

The sensors may seem to be a poor fit to the diagram. This is true if the model is not to show any implementation detail. Instead we could have left the batch to detect when it reached 100oC and switch off the heater and the container to detect when it was full and switch off the filling process. The concept of a domain which is discussed fully below is used to determine which objects should be included.

The objects chosen do not restrict the solution set. They are a statement of the requirements. The objects are essential and abstracted making it possible to choose asolution from total automation to total manual operation.

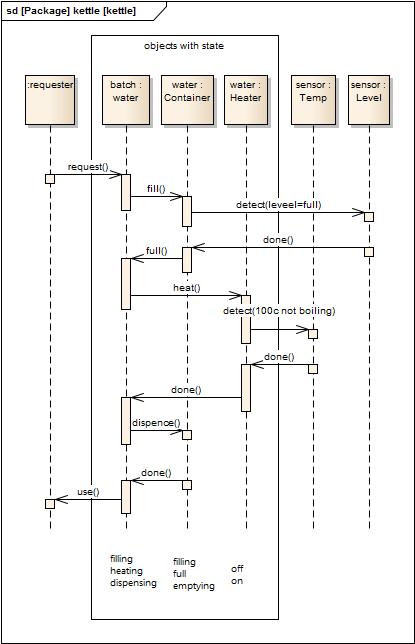


Figure Sequence diagram

The activity diagram shows another way to represent the requirements of the system. The operator activities replace the water batch[[12]](#footnote-12) and sensors. The implementation can use a saucepan and cooker ring for the container and heater.

The solution set is very much restricted. In SysML we use the ‘is allocated to’ relation to impose a solution on the more abstract model shown in the sequence diagram. For example the sensor operation ‘is allocated to’ the operator.

In a similar way the architectural components such as container 1.7 litre with push botton lid and filter, detachable electrical heater and illuminated on/off switch can be used in the allocation for a cordless kettle.

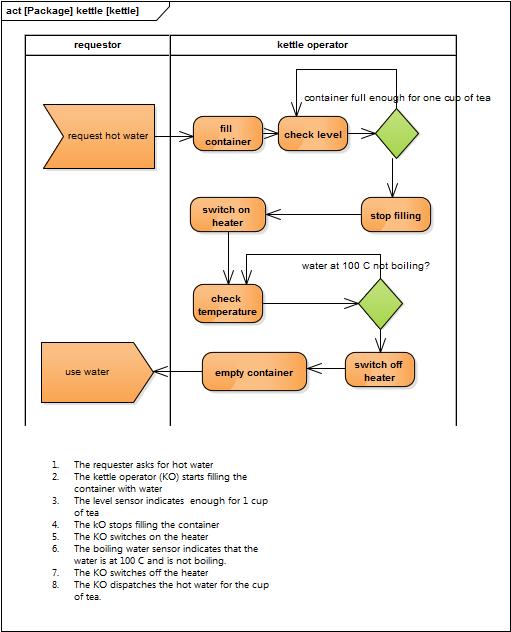


Figure Activity Diagram

The next section discuses the terms solution and requirement in more detail.

# Review of Requirement and Solution Terms

To appreciate the significance of a SysML model an understanding of the difference between a requirement and a solution is needed. System engineeringhas always been based on requirement analysis as the essential process for choosing the best solution. SysML model needs to have requirement statements produced for every level of the Model hierarchy[[13]](#footnote-13). Most often these are produced before modelling commences. These may be imported into the model[[14]](#footnote-14) and traced to all model elements to demonstrate to stakeholders the requirements coverage in the model.[[15]](#footnote-15) Some of these statements will be derived; that is they come about when deciding on a solution. For models with a hierarchy the term solution and requirement are really the same thing. A solution specified using SysML may have a requirement specified using SysML one level above the level of the solution.

There are some absolute truths concerning requirements and solutions.

1. For every requirement there can be one or more solutions. This means that a requirement can be viewed as a solution set. The set size will be governed by any constraints specified in non functional requirements.
2. There must be an abrupt change of subject matter between the requirement and its solution. Given a solution it is not possible to determine the requirement which gave rise to the solution. Given a requirement it is possible to determine if a particular solution fits the set. Sometimes in a practical situation it is hard to get this separation. This is often the cause of over complex, difficult to understand models.
3. Requirements are hierarchical. Given a requirement a number of derived requirements can be created. This is the possible solution for that requirement. As a result the SysML model simultaneously represents the solution and requirement at that level. That is the level below is the solution of the requirement one level above. The top of the hierarchy is the application and the levels below are the set of service domains which are deemed sufficient for the purposes of requirement description. The real physical world objects by definition are not the requirements of the system as they lie outside the model. These objects do not exist until the system is built.

The words abstraction and essential modelling are closely related to the concept of a requirement and its solutions. An abstract object for example represents a requirement which has an associated set of solutions. An essential object is an abstracted object representing a set of requirements to be included.

In the next section the above truths are used to create a divide and conquer strategy which enables all aspects of a complex system to be modelled.

# Divide and Conquer Strategies

There are two popular ways of breaking up a system. Separation into different subject matters called domains and functionally decomposing a system into subsystems. There are pro’s and con’s for each. Functional decomposition is relatively easy to do but it is not a rigorous requirement analysis methodology. It works well in situations when a system of systems needs to be divided into a number of separate systems and subsystems. Requirement tracing and coverage may be poor. Separating out the different subject matters in a system is a rigorous analysis method which leads to a high level of understanding, tracing and coverage. The disadvantages are:- it’s difficult to do, costly and requires the specialist know how of SME’s.[[16]](#footnote-16)

Splitting a system into separate subject matters or concerns has been around in software development for a long time. The layered architectures of the OSI model and SOA are examples.[[17]](#footnote-17) Design patterns have also arisen from the need to decouple the different parts of the software system[[18]](#footnote-18). However for SysML and MBSE approaches this divide and conquer strategy is rarely mentioned. One of the big advantages in a software system is that the separate parts will not be coupled or connected in anyway. This makes maintenance cheaper. The same is true in SysML projects. A model consisting of separate domains is easier to understand and maintain. A greater ROI is possible. The main disadvantage, when splitting into different subject matters compared to using tightly coupled functionally decomposed modules[[19]](#footnote-19), is that projects need to exist for a long time. Only legacy systems achieve this and these would not have been started using SysML. The increasing pace of development is also causing new projects to be delivered in a shorter time span.

One advantage of division by subject matter is that requirements can be more easily traced. Requirements expressed in model form have been rigorously analysed and made more understandable than the original written statements[[20]](#footnote-20). The ease of traceability is a measure of goodness for a model.

The purpose of a SysML model is to provide a specification for implementing a system. A SysML model is a definitive and unambiguous statement of the requirements. It stands to reason that implementers must understand the SysML model. The stakeholders must also be able to understand the model but only to a degree. Requirements in written form can be imported and traced to model elements to demonstrate that stateholder concerns have been addressed.

The next section discusses how models are constructed using separation into subject matters.

# Domains

The meaning of the word Subject Matter is the same as when applied to different study subjects for example geography and history. In a typical system there may be many different subject matters making up the domains of the system. The domain with the most recognizable system intention and purpose is referred to as the application. Other domains, in direct support of the application, are called service domains.[[21]](#footnote-21) Service domains arise because of the need to find solutions to problems arising from how to implement the application and service domains. For example the transporter was implemented as a robot giving the need for a robotics service.[[22]](#footnote-22) The robot may need to accept commands from a person, for example start and stop, generating a need to interface a person to the computer. This is the MMI service.and practically every system has one. For example to open the doors on a train a passenger is obliged to press a button when the train stops at a station. The application contains a door object and the MMI service domain contains a door object as well which remembers if the door is open or closed as part of the interfacing to the operator. In many systems resources are shared so there needs to a contention resolution service domain.[[23]](#footnote-23)

The SysML model can be thought of as the description of topics (subject matters) which enables further supporting topics to be discovered. The topic description or domain description is a set of requirements placed on lower level supporting domains.[[24]](#footnote-24) One is at liberty to design/choose which service domains to deploy and through requirement analysis, i.e. constructing the domain model, derive further domains and requirements.

Domains should include written descriptions of its purpose and responsibilities. Included will be what the domain knows. For example this domain knows how to transport heavy objects.

All objects[[25]](#footnote-25) which make up the subject matter of the domain must have a name, purpose, and responsibilities. Having a name is not enough as names may be the same in different domains representing the different roles of the same physical object.

The next section shows how domains can be derived from the application domain.

# How to find the service domains of a system

Service domains are determined by how the application domain is to be implemented. The concept of allocation, discussed below, is used to specify implementation domains. For example, in the section above, the implementation for signals found in the application are allocated to an MMI domain and transporter movement implementations are allocated to a robot. These two service domains were discovered because a way of interfacing a human being to a computer system and a method for transporting heavy machinery in a constricted area were needed.[[26]](#footnote-26)

However not all domains can be found in this manner. The need for a fault analysis domain would be identified from requirements specified by stakeholders. It would not be possible to deduce this from the application domain alone.

One important class of service domains which cannot be derived from the application is the Architecture domain. Architectural domains are very important. The next section discusses them in more detail.

# The Architectural Domain

Most SysML projects will include a model architecture template, which specifies how the model is to be constructed. The SysML model will also show views of the system architecture which is the primary concern of the System Architect.[[27]](#footnote-27)

For example dividing the model up into domains is an example of an architectural domain applied to the model. An example of system architecture would be a view of FADEC control showing the safety aspects introduced into the processing units using duplication of the ECU and sensors. Another example of system architecture would show how each engine parameter determination was safe guarded through independent determination.[[28]](#footnote-28) This system architecture could be modelled in a separate domain dealing with components of the control system that need duplicating and monitoring.

Architecture can be thought of as a set requirements which specify how a system or model (or for that matter anything) is to be constructed.

Dividing the model up into domains is an example of an architectural domain applied to the the model. It is one of many possible conquer and divide strategies which can be deployed to show how the model is to be constructed.[[29]](#footnote-29) This write up, which is advocating the use of domain analysis, is a model architectural strategy that can be applied to the design of real time systems.[[30]](#footnote-30) The terms domain, subject matter, application, service describe how to divide up the system model to make the system design and model construction easier to handle.

In order to clarify more fully what is meant by a domain model it is necessary to explain what is meant by a domain property. In particular the property that measures how logical or physical a domain is. This is the subject of the next section.

# Domain Properties

An important property of a domain is its classification into application and service types. There is another property which is used to specify the degree to which a domain may be thought of as logical or physical. Both behaviour and structure models can feel more physical or more logical depending on what is being modelled. The behavioural model though will often appear to have a logical feel about it; i.e. be abstract, removed from reality, and express only those things which are essential. The logical model should only express those things which are essential and like the behavioural model is purposely devoid of detail to express only those essential properties. The structural model on the other hand has a concrete feel to it, represents the physical world as is and can be full of detail to be useful. Projects which have well documented descriptions of physical hardware will be lead by SME’s.

There are parallels with the world of science where theories are tested to see if the predicted solution sets fit the real world. When Einstein said “make it as simple as possible but no simpler” he was referring to abstraction and to finding those essential constructs necessary to make an abstract model. In particular there are parallels in mathematical modelling which uses hierarchical unfolding of solution sets to predict specific outcomes. The maths is a set of requirements and derived requirements which specify the behaviour of the real world.[[31]](#footnote-31) The behavioural model for a domain is really the same thing and comes with a similar set of challenges for the analyst.

A physical object name can be used in the model at any level of abstraction regardless of whether it is a structural or behavioural. The terms logical and physical are quite loose as structural diagrams are often regarded as logical. More usually one applies the term to parts of the model stating that this part is more physical or more logical meaning that the domain property physical or logical is a relative term.

The physical components themselves should not be the focus of the SysML effort. Instead work should concentrate on producing a logical model based on the analysis of the system’s domains. This approach will optimise the efforts of the design team to choose the best solution and make it more probable that ROI will be perceived as good by the project managers.

SysML uses the concept of allocation to better define the difference between a domain which is logical in nature and ones which more represents real objects.[[32]](#footnote-32)

In the next section the relationship between the physical and logical worlds is discussed by showing how the logical components of the domain model are allocated to physical components representing the end solution. This exercise provides context for the behavioural model description and aids understanding.

This is an example of just one type of relation between objects in different domains. The next section also discusses synchronisation another type of relationship between domains

# Domain Relationships

This section discusses domain relationships between objects in different domains beginning with the ‘is allocated to’ relation. The other important relationship is synchronisation between objects modelled by ‘is synchronised by/counter parted by’. These two relationships can change in real time as the system behaviour changes.[[33]](#footnote-33)

## SysML concept of Allocation

The general concept of an allocation is to make a model more concrete in order to realise a solution. The ‘is allocated to’ relation is used to connect the logical domain to its more physical implementation. For example adomain model consisting of one application and a series of hierarchical services may be allocated to physical components. These components may be modelled in a structure, for example a block diagram, to show the physical arrangement and the physical flows between components. [[34]](#footnote-34) The logical model may confusingly contain the names of the physical components as well. This closely follows the RUP deployment concept. There is a need to determine which behaviours are carried out by which physical objects.[[35]](#footnote-35) In the transporter example, mentioned in the introduction, the transportation behaviours were allocated to a robot. This gave rise to a robotics service domain whose subject matter is about an ordered sequence of steps using the robot’s instruction set. The allocation means that the behaviours specified in the application domain are to be carried out using the specified robot. The chosen robot may come with a configuration facility so that these behaviours can be implemented or the behaviours are directly allocated to specifically written software for the robot.[[36]](#footnote-36) The logical model may contain logical objects which can be directly allocated to physical objects. For example the transporter can be allocated to a robot or an illuminator can be allocated to a gas light.

Logical to physical allocations can also be visualised using a V model. In this approach the system is modelled from abstractions of the real world objects with the roles played by the physical objects being represented as proxies.[[37]](#footnote-37) This idea is used in control system design architecture discussed below,

Attempting to find and develop the best solution is a ROI goal and can be very much aided by the proper use of allocation diagrams and tables.[[38]](#footnote-38)

To further illustrate the concept of allocation one can consider different solutions for the same modelled requirements which are simultaneously implemented in a system. Many systems can operate in different modes. Furthermore the system is designed to switch modes either automatically or on demand from an operator. For example an automated landing system will have safety overrides which give control back to the pilot. Modes of operation can be selected in an attack radar system which allocates some of the automated functions back to the pilot. A traffic light system may change to a different lane assignment algorithm during rush hour.

Re-allocating the behaviour to different hardware is a common way of dealing with system errors and unexpected behaviour.[[39]](#footnote-39) In a use case model the errors and unexpected behaviour is specified as success conditions and minimum guarantees. (see References Use Cases)

Allocation is a way of realising the solution to the set of requirements specified in an application or service domain. Allocations create implementation specifications which when modelled represent sets of requirements as to how the system is to be realised.

Allocations to hardware give rise to further service domains. In a typical system these domains must be kept in step or synchronised in order to achieve correct system behaviour.

Domain synchronisation specification is the subject of the next section.

## Synchronisation Specification

These are the synchronization relationships that are needed to keep the state[[40]](#footnote-40) of each domain in step with changing external events and other domain state changes.

The definition of a domain as a separate real, hypothetical or abstract world[[41]](#footnote-41) ensures that changes in the definition of the domain do not cause changes in the definition of another.[[42]](#footnote-42) E ach domain instance will need to reflect the real world state as it changes or if one domain is relying on another, instance data changes in the first will need to be reflected in the second. For domain instance data there is an domain independent synchronisation process keeping the domain state in step with its dependent domains and the where necessary the world state.

This synchronisation process is specified by the ‘is counter parted by relation’. If two domains collaborate a change in the state of one object in one domain will cause a state change in one or more objects in the second domain. This can be modelled in a mapping rule.

The nature of this relationship can be demonstrated using the C++ and assembler computer languages. A C++ Compiler converts C++ code into Assembler using a set of built in mapping rules.[[43]](#footnote-43) The compiler is not a converter but a synchronisation process keeping the assembler code in step with the C++ code as it is edited and changed. This means that changes in the C++ code produce corresponding changes in assembly code by a mechanism that has nothing to do with either assembly or C++. Another example along the same lines as above would be the German to English language translator and in the transporter example the routes specified in the application domain is synchronised with the robotic instruction sets.

In the runtime system the service and application domain(s) are synchronised with changes that occur in real time by events which come from outside the system. For example when an aeroplane takes off the “weight on wheels” event signifies that the plane state has changed from being on the ground to being in the air. This in turn causes the wheel retraction system to change from being fully open to fully retracted. Signals are sent to the retraction hardware system to retract the wheels. This process has nothing to do with the aeroplane application domain or the retraction system domain. It is an implementation of the synchronisation specification. The specification not only lists the counter parted objects but also the signals that leave the first domain and the counter parted signals that enter the second domain.

These signals form the interfaces between domains and are also known as input / output devices or adapters. Inside the domain the use of sequence diagrams is used to specify how the objects are synchronised when receiving signals from another domain. [[44]](#footnote-44) In communication systems the protocol stack provides the means of implementing synchronisation between geographically distributed elements. The protocol stack is specified in standards used worldwide.[[45]](#footnote-45)

Domains may also be synchronised on a continuous basis. For example temperature readout is synchronised with its device reading.

An example of a synchronisation implementation in software is the procedure call. Information from the first domain is converted into types compatible with the types in the second domain before the procedure is called by the first domain. This can be used in both continuous and event driven systems.

An example of synchronisation implementation using only hardware would be the German English electric adapter plug.

All synchronisation specifications have to be implemented. Some become part of the run time system and others are stand alone.[[46]](#footnote-46)

The next section looks at how a control system may be designed using domain synchronisation and allocation.

# The Control System Model

Sooner or later in a real time system one is faced with the problem of designing the system by deciding which physical objects will need to be tracked and controlled. The design can be shown in a control system model. One of the best OO approaches to this can still be found in the methodology first proposed in the book Object lifecycles by Shlaer/Mellor.[[47]](#footnote-47)

The concepts of synchronisation and allocation with the notion that a requirement is a solution set can be used to make sense of a model containing abstractions of real world objects. These abstractions can be modelled as objects in activity diagrams with swim lanes and in sequence diagrams.[[48]](#footnote-48) State machines can be used to show the dynamic behaviour. By introducing a physical abstraction the solution set is reduced from enormous to something manageable.[[49]](#footnote-49) A telephone system might contain a telephone line. A traffic light system might contain a road or lane abstraction. The transportation system might contain a transporter abstraction.

The physical object needs to have specific behaviours allocated as shown for example by an activity diagram with swim lanes. These behaviours are the control objects.. A control object needs to be synchronised with the physical object it is tracking and controlling.[[50]](#footnote-50) The control object serves as an abstraction of the physical object, is the dynamic requirement and solution set for that object.

In an object orientated approach to control system design, the control objects represent the behaviour of the corresponding physical object. The control system model is often mistaken for the complete system model when in fact it only represents those behavioural requirements allocated to the physical objects which are specified in the design process. The control object is also the interface facade for the physical object. In practise the actual physical object may be completely unrecognisable from the object described by the facade,[[51]](#footnote-51)

A single physical object may play only one role in a system or many roles. The different roles may be in different domains, A control object only represents one role of the physical object.[[52]](#footnote-52)

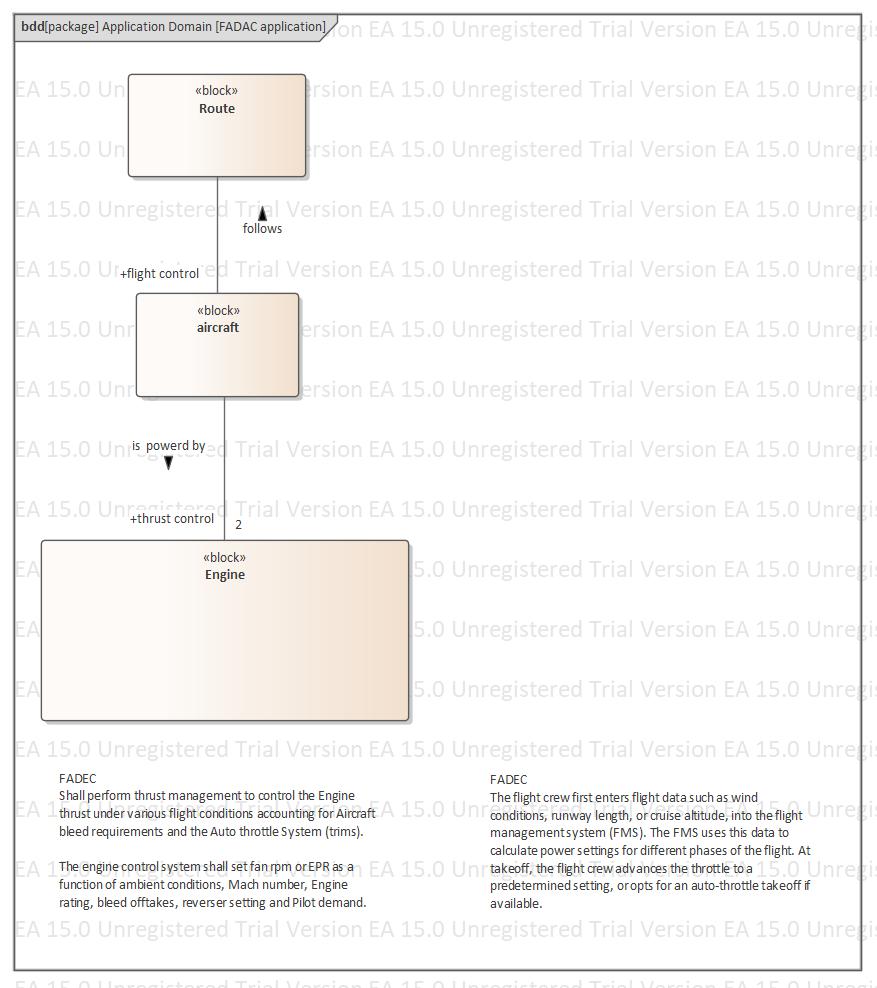
The control objects are proxies for the real objects roles of the system design. A diagram can be drawn showing the control relationship between the proxy and the real object. This is a many to many relationship. A single proxy can control one or more physical objects. A physical object may be controlled by one or more proxies. Where a proxy controls more than one object its acts as a co-ordinator for that set of objects. If the physical object has more than one proxy then each proxy represents a role that the physical object plays in the system. Usually in a control system model the architecture will dictate that the co-ordinating proxy try to delegate responsibilities down to individual proxies representing a single physical object role. Control object behaviour may be allocated in part to a human operator depending on the degree of automation.. The operator is prompted by the system when operator input is required.[[53]](#footnote-53) Proxies that have the most context knowledge are often associated with operators. Other proxies with less knowledge are used to control objects which have a more re-usable feel about them. For example a temperature regulator or an robotic arm.Aco-ordinating proxy is in-directly associated with physical objects. Examples include call control in a communication system, batch process in manufacturing thrust control in a jet engine, access control for a car park. If a physical object has more than one role then theses roles may be in a different or same domain.

The architecture of the control system conforms to the following points.

1. A control object is always a proxy for a physical object or a set of physical objects.[[54]](#footnote-54)
2. A proxy is a role in the physical model.[[55]](#footnote-55)
3. A proxy is the same as a control object’s whose purpose is to keep track of and control physical objects.
4. The proxy and the corresponding real object are really a single object. The proxy may be implemented within its physical counterpart as in a robot.
5. The proxy and the real physical object must be synchronised.
6. The interface/facade between the proxy and the external equipment may give an illusion as to the nature of the physical hardware being tracked and controlled. In practise the real physical object may be unrecognisable and invisible.
7. The physical object is deployed during system design and solution choice.
8. The control system is implemented by allocating the control elements (proxies) to hardware, software or peopleware[[56]](#footnote-56)[[57]](#footnote-57). The logical and physical architecture dictates the arrangement of the elements. This can be physically centralised or distributed across any number of hardware platforms.

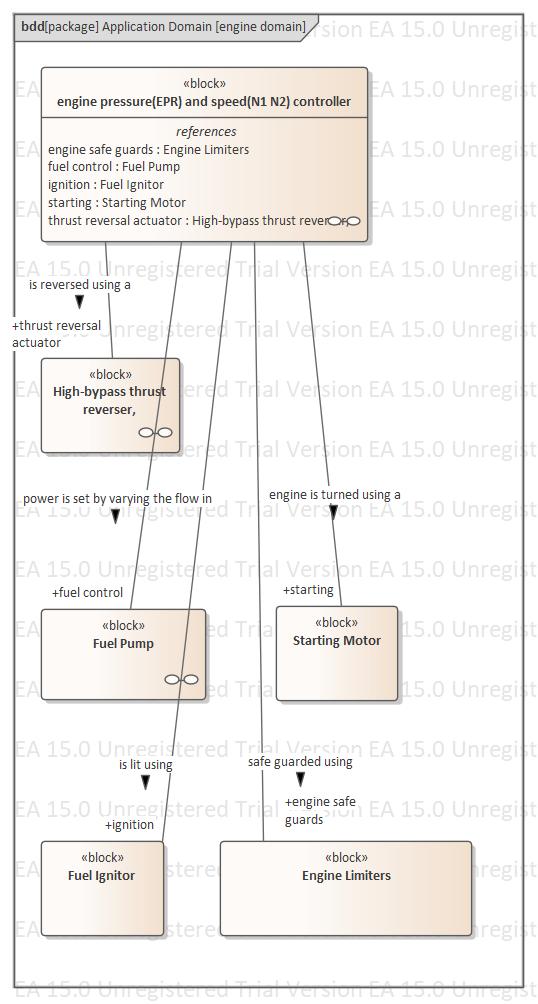
## Entity Relation Model

It becomes important to be able to identify the physical object roles which is pertinent to the design of the control system. In the diagram shown below, the role of the engine to develop thrust is shown by the dynamic relation “thrust control” role between the engine and the aircraft. A jet engine develops thrust according to the setting of the throttle set by the pilot. The thrust settings may be pre-determined by the phases of the flight as in the FADEC. The two proxies “flight control” and “thrust control” specify the behaviour of the FADEC.



## Figure FADEC operation

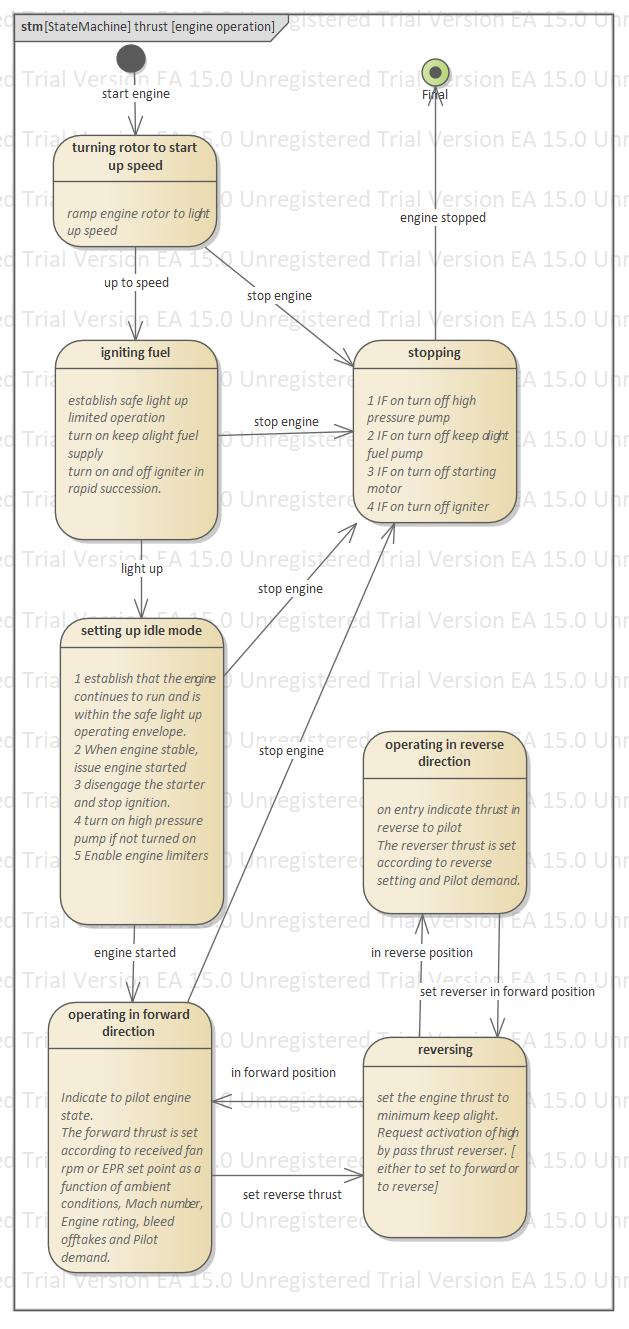
The next diagram shows the details of the thrust control proxy shown in the previous diagram. This is a new domain the engine domain. Here thrust demand is met by altering the flow of fuel to the engine while measuring the EPR, N1, N2 parameters At the same time safe guards are applied to ensure that the safety operating envelope is adhered to.The thrust demand is converted to EPR,NI,N2 at the boundary between the two domains. In addition the FADEC flight and thrust control proxies are synchronised with the proxies in the engine domain.



## Figure 6 Engine control Domain

## State Machine

The next diagram shows the state machine for the proxy pressure and speed controller in the ERD “engine domain”. The state machine has two principle operating modes forward and reverse. Note that some functionality would be partly allocated to the pilot depending on the selected operating mode.[[58]](#footnote-58)



## Figure 7 showing engine speed and pressure controller state machine

## The V Model

A model using abstractions of real world objects is the OO approach for system design. A context diagram[[59]](#footnote-59) in the physical model shows the scope of the control system. For example a block diagram would show the control block with the dependant physical objects connected to it. The V model shows the same information in a hierarchical view.[[60]](#footnote-60) On the left side the physical objects are arranged in context order, the larger the context the further up the V the object is placed. The corresponding proxies are shown on the right again in context order with the highest at the top of the V formation.[[61]](#footnote-61) This ordering gives a clue as to how delegation plays a part in the design of a control system. For example in the diagram below the flight control delegates to the thrust control to control the thrust under various flight conditions.[[62]](#footnote-62)

A context/V model is useful for showing physical objects and their proxy counterparts across domains. The engine limiters and fuel pump is not in the same domain as the aircraft and engine.

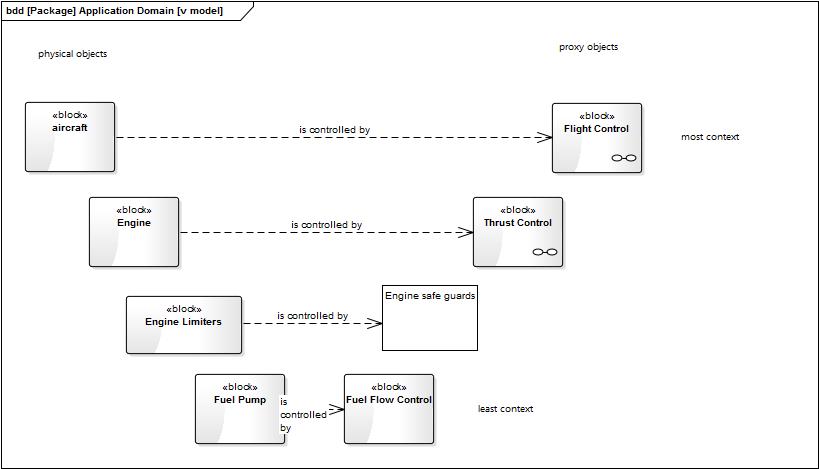


Figure 8 Engine Control V model

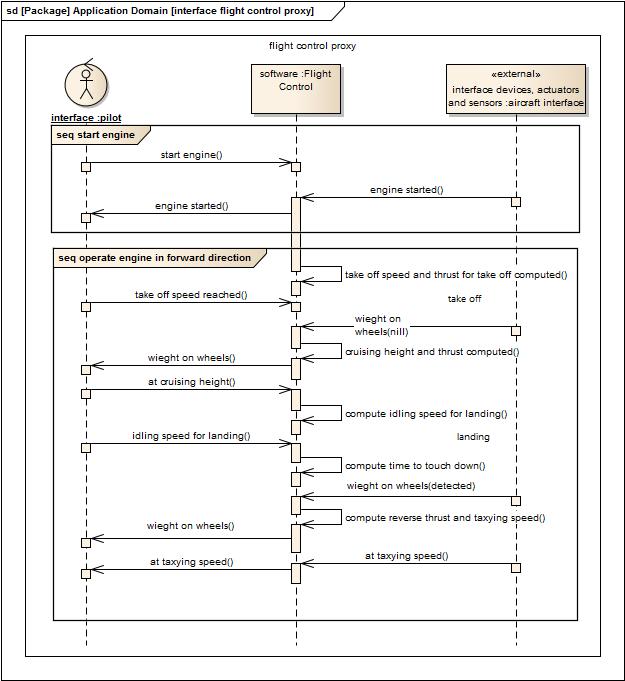
## The Interface Model

The control object/proxy will need to interface to the physical hardware it is tracking and controlling. Even if the proxy is a co-ordinator and only indirectly controlling physical hardware it will still need to interface to external objects. The flight control proxy, which is controlling the aircraft along its route, is implemented by allocating the flight control behaviour to software and to the pilot. The pilot and the executing software will need to be synchronised. In addition the flight control software will need the status of the aircraft. This is shown in the diagram below.

Allocation to Liveware, Hardware and Software of all identified proxies allows the control system to be designed. It is at this stage that the logical architecture can be introduced. The V model suggests an architecture that is layered so that context rich information comes at the top and the reusable service orientated parts at the bottom.[[63]](#footnote-63) The first layer represents user input and output via an operator interface. The middle layer usually implemented in software makes up the bulk of the control system functionality. The bottom layer is the devices layer interface and the sensors and actuators layer.

The sequence diagram shown below follows this architecture and shows the interface to the aircraft and pilot as being part of the flight control proxy implementation. The interfaces are the synchronisation implementation between the control domain, pilot services and the sensors and actuators service domain.

A common mistake in modelling is to imagine that the sequence diagram is showing the dynamic behaviour of collaborating objects. In fact the sequence diagram shows only the synchronisation specification between domains and between objects in the same domain.[[64]](#footnote-64) Sequence diagrams and activity diagrams can be converted/compiled into a run time environment. To demonstrate dynamic behaviour other tools are used such as Matlab and Simulink.

Figure 9 flight control proxy interface

## The Continuous Process Model

The above examples have been largely event driven to illustrate control system design. Events are associated with states of the system and how those states change. When in a state it is typical to have continuous processes running. In SysML continuous processes are modelled using parametrics.This is demonstrated in the next two examples below.

#### Example 1 Robot Lifting Arm

A steady state velocity schedule is used to control the speed of the lifting arm when the arm is not accelerating or de-accelerating. This service would be equally applicable in all lifting processes for example in a lift system. The control system would be modelled first using a variety diagrams for example, state machine, sequence diagram or activity diagram. To show the continuous process the states of the dynamic objects would be found first. The diagram below shows the transportation and lift tasks modelled as two properties of the control block in a internal block diagram.[[65]](#footnote-65)

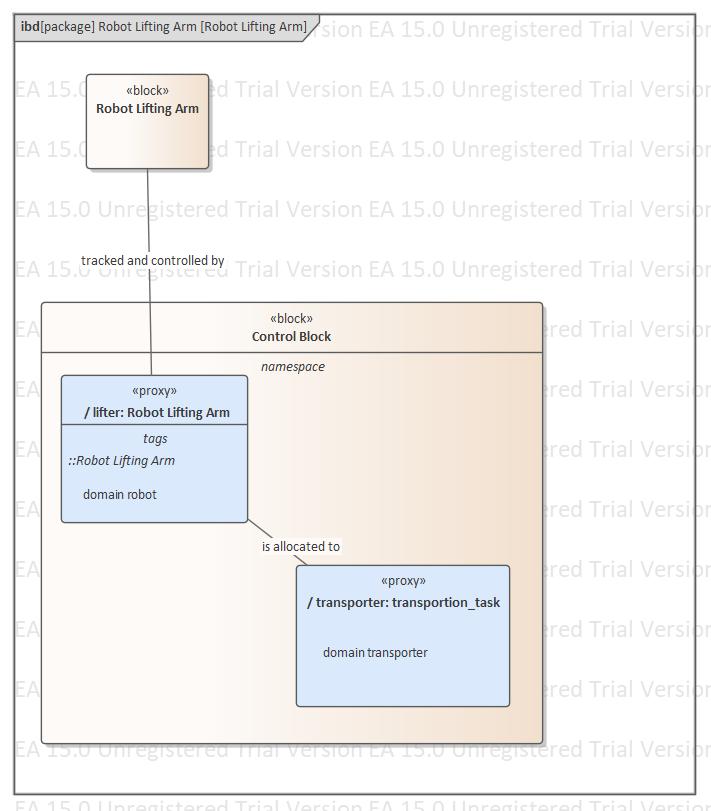


Figure 10 roles lift and transportation task

The Robot lifting arm proxy has two states idle and lifting shown in the diagram below. In the lifting state the robot will be raising and lowering the load in synch with the transportation task.In the lifting state the steady state velocity schedule would be deployed to control the speed of the lift.

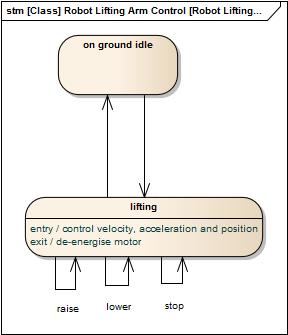


Figure 11 continuous processes in the lifting stateThe steady state velocity schedule keeps the velocity of the lift arm constant during the lift. It uses a feedback loop and velocity set point characterised by the velocity schedule and velocity to power equation as shown in the parametric diagram below.

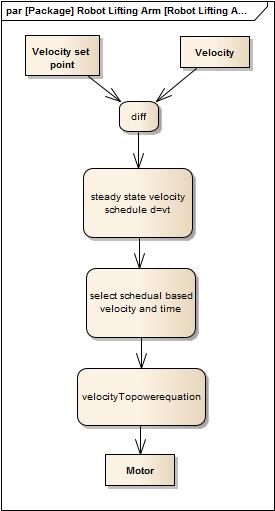


Figure 12 Steady State Velocity Schedule

#### Example 2 Fadec

The FADEC is the name given to modern practise in jet engine control where controlling all aspects of the aircraft engine performance are carefully considered particularly the safe control of engine thrust.

Once the engine has entered its operating state, thrust on demand is produced depending on the throttle position, bleed off takes and other factors as shown in the state “operating in forward direction”. (Figure 7 showing engine speed and pressure controller state machine). The engine safe guarding ensures that the engine operates within a multi-dimensional safety envelope[[66]](#footnote-66) by constraining the engine performance. This is a continuous process best modelled by the use of a SysML parametric diagram [[67]](#footnote-67). An example diagram is shown below. The engine parameter EPR set point is used to compute the feul flow demand. A feed back loop of the measured actual EPR ensures that the engine transcients are accurately smoothed to provide the necessary engine performance. Exceedances of speed or temperature use the engine limiters which are switched in instead of the EPR schedual.

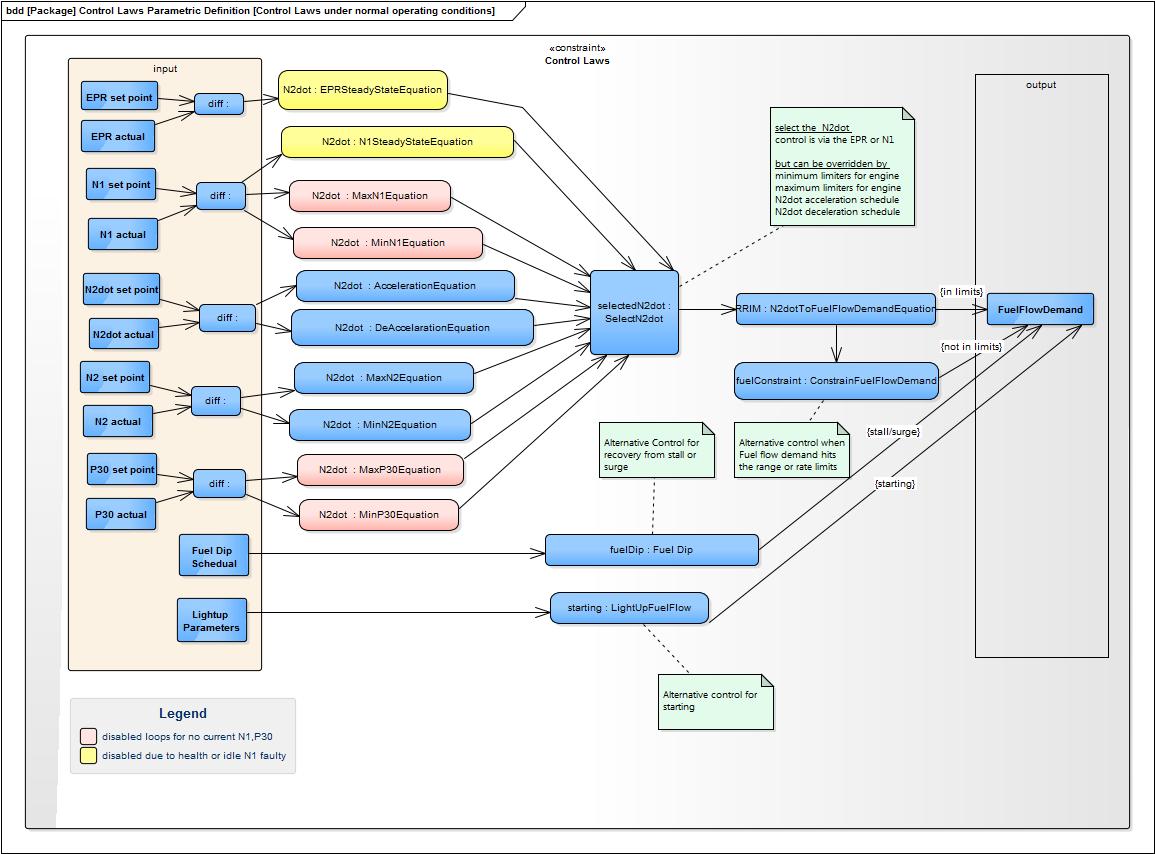


Figure 13 FADEC

## Further Entity Relation Diagram Examples

#### A Train Door

A real time system responds to events driven from external world. The physical objects in the environment are synchronised with the control systems proxies implementing the system. The train driver enables the door locking mechanism so that the door can be opened by a passenger allowing access from the train to the platform.[[68]](#footnote-68) The passenger and train driver are allocated to behaviours identified in the model shown below. The role “passenger access control” is responsible for creating and deleting passenger access to the platform. Access is created depending on the train state and passenger presence. Passenger access control proxy which is modelled as a role of the train delegates to the carriage access control which delegates to door control. Carriage access control determines which doors can be safely opened. Door control opens and closes the door.

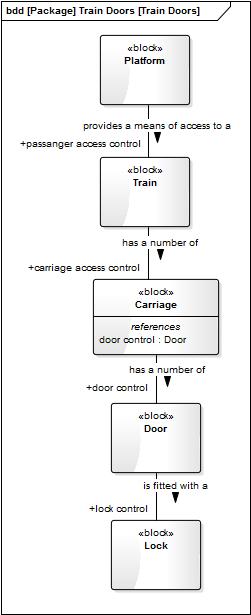


Figure 14 Train Door Operation

#### A Telephone System

A telephone system consists of a number of subscribers or subscriber end points who establish a call using the switched pool resource. The roles ‘+connection’ and ‘+party involvement’ make up the proxy objects of the control system for the application domain. The call can have more than two subscribers and a subscriber can be involved in more than one call.

This example illustrates application and service domains. Synchronising between domains and between objects in the same domain requires very complex, standardised communication protocols because the equipment is widely physically distributed and different manufactures make the equipment. Protocols can be modelled quite effectively using a state machine. The “V” model can be used to show the sender and receiver blocks. Protocols sit well down in the “V” model and have little system context.

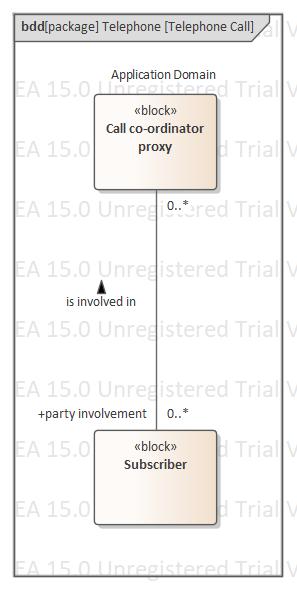


Figure 15 Subscriber Call

The implementation for call handling can be abstracted using the concept of a switch. The mechanical switch consisted of a metal bar and manually operated switches to be able to connect the subscriber lines together. Modern switches use multiplexed digital transmission. +switching is the role of the switch when making connections in a n party call. The switch pool contains a number of switch resources which are assigned when the call proxy reaches a certain state. The call handling application domain needs to be synchronised with the switching domain service.

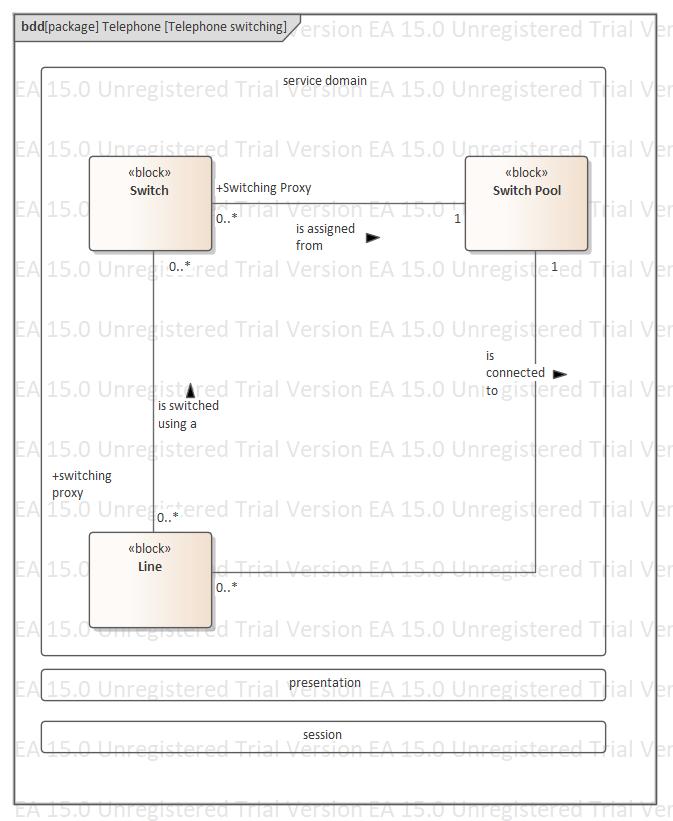


Figure 16 Call Switching

This service domain is shown below. When allocating resources in a real time system the problem of contention will arise. This can be resolved by using a ‘first in first out’ strategy[[69]](#footnote-69) but many other strategies can be deployed. The use of AI in this area is common. The block switch rquest proxy could be serviced in time order by the switch assigner proxy as shown in the diagram below for example.

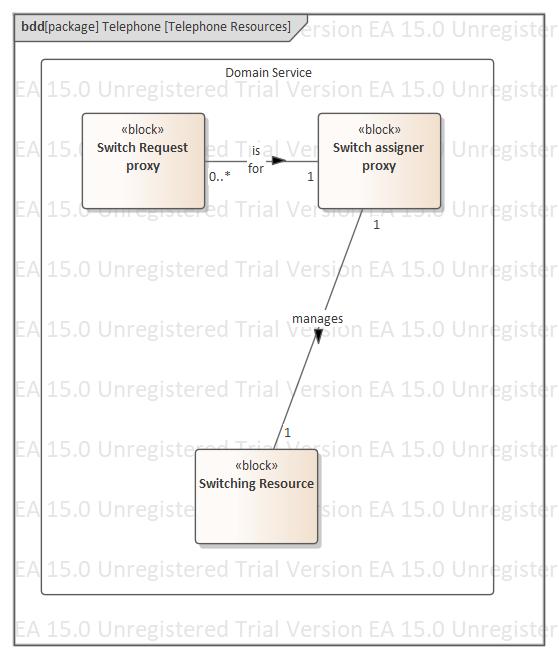


Figure 17 Telephone Resouces Management

#### Lift

In this example the lift system consists of one or more lifts serving a number of floors. The +request, +lift control, +lift access control proxy’s make up the main control elements. A request can be initiated from a floor to call a lift going up or going down. Once in the lift the destination floors can be selected. The lift will serve the pending stops list by serving all the up requests before changing direction for the down requests and vice-versa. When a lift stops at a floor it will remove all pending stops for that floor. When a lift has served all stops it will return to a holding position.

The lift conflict[[70]](#footnote-70) which can arise when a request is initiated from a floor is resolved by a separate service domain which when synchronised creates stops and decides which lift to move in the lift application domain. This domain is similar to the resources assignment shown in the telephone system.

The lift movements between floors would be typically controlled using a constant velocity schedule shown in Figure 12.

Opening and closing the doors is controlled from the proxy +lift access control and is similar to Figure 14 Train Door Operation

The lift will typical know which floor to serve next and the direction of travel. This may also be shown as relationships between lift and floor and lift and lift pool.

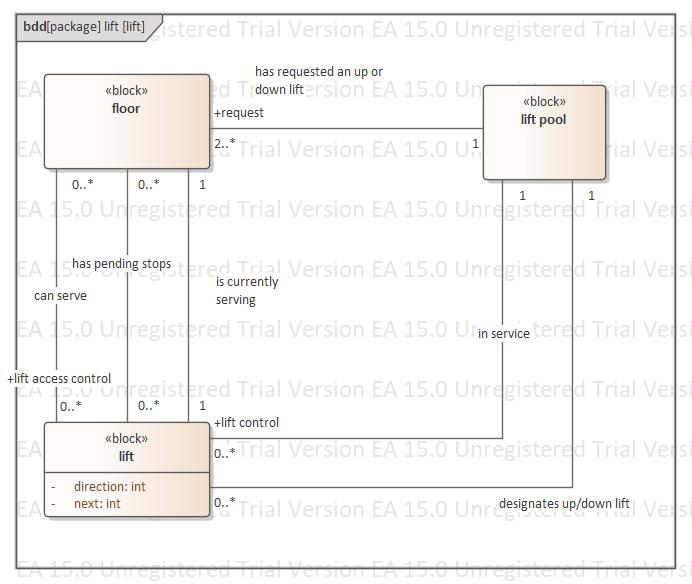


Figure 18 Lift System

#### Traffic Lights

The traffic light system represents the behaviours associated with a road junction or cross roads. The road system consists of a number of traffic flows which use routes composed of shared segments which are assigned to the routes as they become available. The model also covers rounderabouts with or without traffic lights, pedestrian crossings and one way system. The control system is made up of +traffic flow, +route flow, +route segment flow proxies. A service domain similar to the assignment problem of lifts to floors schedules the traffic flow along configured routes without contention. Routes are made up of ordered segments which are shared. The scheduler may allow more than one traffic flow simultaneously. The scheduler service is synchronised with the traffic light system application service allowing traffic flows to start and stop. The +traffic flow proxy delegates to the +route flow proxy which delegates to the route segment flow proxy to control the traffic flow.

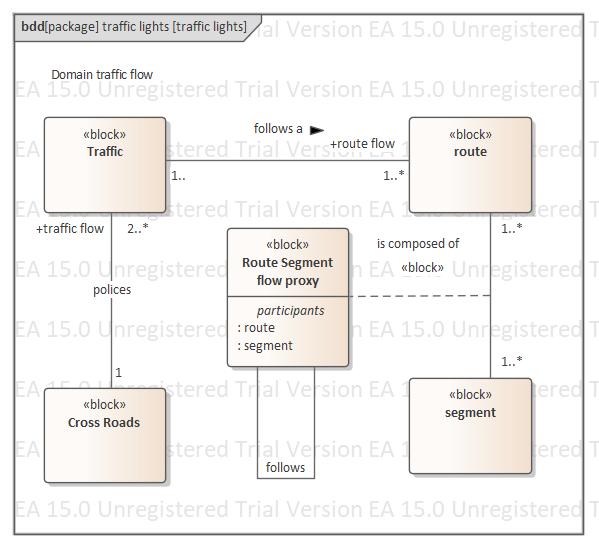
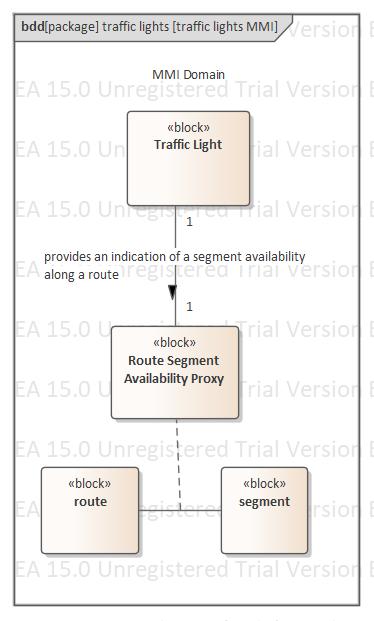
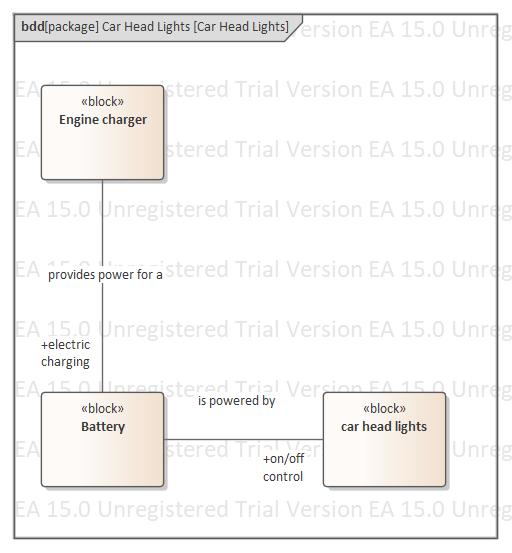


Figure 19 Traffic Light SystemThe diagram belows shows the MMI domain for the traffic light system.[[71]](#footnote-71) A traffic light physical object is controlled by the route segment availability proxy. Where segments become simultaneous available the number of physical traffic lights needed can be reduced. The traffic light shown in the diagram is therefore logical traffic light. In an implementation typically a traffic light would be placed at the beginning of the route and not at every segment. The missing traffic lights are really implemented in the drivers mind. This shows the importance of allocating functionality fully including to any human operators.



#### Car Head Lights

This system demonstrates the feature found in most modern cars to turn off the headlights if left on after the engine has been switched off. The implementations can be allocated[[72]](#footnote-72) to cover the scenarios driver warning and automated solution. The implementation driver warning requires that the role of on/off control of the headlights is allocated to the human driver. This requires a human operator interface service domain.. The charger is made responsible for keeping the battery charged. This is done by the proxy +electic charging. When event engine off occurs the proxy will turn off via the proxy +on/off control any appliances which are still on.



# Summary

The transport system can be used to illustrate the main concepts introduced in this write up. Two domains: the transport system as the application and the robotics as the service domain were identified. These served as our divide and conquer strategy and also the system architecture. The responsibility for controlling movement was allocated to a robot which identified a new domain concerned with the execution of a series of movement instructions. The routes specified in the application had to be synchronised (compiled) with step sequences executed by the robot to create the right transporter movements. The transporter was identified as a logical object which was used in the logical model to allocate behaviour to the physical object robot and to demonstrate the system’s operating modes[[73]](#footnote-73). The process of domain analysis produced a set of requirements in model form which allowed solutions to be defined. This domain analysis made certain that the robotics domain was not the same subject matter as the transporter domain by creating an independent movement instruction set for the robot.

The control system design of a typical real time system was discussed. The discussion followed a route that showed that the design could be derived from the physical objects making up the system and the roles they played. These roles were counter parted with proxies who were then allocated to software hardware or peopleware.

This article attempts to explain why a SysML model should be regarded as a requirement document. Projects which by pass the logical model development, and go straight into component design risk losing the usefulness of SysML . SysML should be regarded as an essential tool with which to analyse and firm up the system requirements making it possible to produce the correct solution.

# Feedback

(For improvement suggestions or different views)

Email [terence.buckle@btinternet.com](mailto:terence.buckle@btinternet.com)

# References

(List of books/articles which I found most applicable)

Object lifecycles shlaer/mellor

Booch OO analysis and design

Ivar Jacobson Object-Oriented Software Engineering

James Robertson Requirement Process

Design Patterns Gang of 4

Moreland Artisan Software

SOA Thomas Erl

SysML Distilled Lenny Delligatti

The design of designs Fred Brooks

Writing Effective Use Cases Alistair Cockburn

# Notes

Depending on feedback there is an intention to improve this article.

# Issues Resolved

1. ~~The SysML concept of allocation can be used to model the Hardware components representing the physical world. All other domain models are deemed logical and represent the requirements of the system. Should the physical components only be shown in model allocation diagrams as by definition they are perceived as non logical. A different approach is indicated in number 13.~~
2. ~~The relationships which occur between service/application domains were illustrated by the C++ compiler example. Is this type of relationship generally true? The relationship between domains can only be one of allocation or synchronisation. These concepts are used to show that one domain may be compiled or translated.~~
3. ~~In the transporter example, the write up suggests, that the robot domain can only be found after an allocation is made. Is this true.~~ ~~No service domains can be designed to satisfy the requirements specified in another service or application domain~~
4. *~~The logical models are described as a set of requirements - but explain the difference between this and plain sentence statements~~. ~~There is no difference. The requirements stated in sentence form should be imported into the model and converted to model form. Tracing should be applied. Any extra derived requirements shown in the model should be reconverted to sentence form before further design takes place.~~*
5. *~~Add another example rear view in a car. Mirror or camera design.NOT DONE~~*
6. *~~Treat control system as a special case. Is it a special domain is it OO or what. Fit it in with the above description.~~ ~~Control system is a set of proxies representing the behaviour of objects which make up the system. Entities within the control system must track and control those objects. The control system need not be just software. For example a governor on an engine maybe a centripetal device or a feedback loop and sensor.~~*
7. *~~Objects in the logical model must be allocated to both the hardware and the ECU component(s). The Sysml model is not just software but is composed after allocation of hardware, software, firmware and liveware.~~ ~~The sysml model/domain will consist software, hardware and firmware as shown by the allocation scheme which maps primarily logical entities to physical entities in different domains.~~*
8. *Too many footnotes*
9. *~~Explain resource allocation in real time systems~~*
10. *~~The application domain provides services only to human beings. The beneficiary is human. But be wary of the human who is Part of the system. (Usually a service provided but the beneficiary just has that benefit but does not further process it. A single person may also have many roles.~~*
11. *~~Treat interfaces as the synchronisation required when data is sent form one domain to another.~~*
12. *~~Logical and physical worlds are just examples of domain types~~*
13. *~~Treat logical physical and abstraction as properties of the domain. Treat allocation as a relationship between entities in two domain which are characterised as being logical and physical.~~*
14. *~~Section names~~*

*~~Domains~~*

*~~Domain properties~~*

*~~Allocating domain entities~~*

*~~Domain Synchronisation : between control system and external hardware~~*

*~~Designing the control system~~*

1. *~~Control system example. Robot lifting arm proxy and hardware for the robot arm~~*

*~~The proxy computes the torque necessary to raise the lifting arm vertically carrying a load to a given height. A constraints service domain computes the toque continuously as the load is raised. The torque value is used to calculate the dc voltage applied to the DC motor. Loads up to 10 tons can be raised.~~*

*~~A constraints domain uses parametric equations s=1/2at2, s=vt, s=s3-1/2at2~~*

*~~Control system hardware objects play different roles in different domain. Beware of names and naming of the same object in different domains as this can lead to confusion as to the purpose of the domain. Think of a hardware object for example lift arm as playing different roles in different domains and name them differently. For example lift arm in the application to lift object s off the ground before moving them. Position controller which can position to XYZ coordinates. The latter is a service domain which can in principle be used in many different applications.~~*

*~~The control system write up is confused. Best to start with application domain as an example. Then show how the implementation leads to service domains and the concept of external hardware mirroring the internal objects.~~*

*~~Design must start from the basis that the external objects are not known initially. Modelled using an activity diagram without swimlanes.~~*

*~~The context diagram model is a separate domain showing how the external hardware is synchronised with its proxies.~~*

1. *~~Splitting into domains is a form of requirement analysis~~*
2. *~~When trying to divide a complicated confused subject matter into separate domains one should approach this with a degree of pragmatism. Most projects are commercially driven so spending a large amount of time on this type requirement analysis must be justified and costed out. At the end of day there will always be a sense of incompletion. Apply the 80% 20% rule to this activity and seek out the advice of the project manager.~~*
3. *~~Domains must include its purpose and responsibilities. Included will be descriptions of what the domain knows. E.g start this domain knows how to … This needs explaining e.g. thrust is responsible for insuring that the thrust is not reversed when the engine is not in safe mode. This means basically idling and on the ground. The engine must be idling when the reverser is activated for forward and reverse modes. The engine must ensure that exceedences are adhered to. This means the limiters are switched on when the engine is started. Not separate limiters for reverse and forward thrust.~~*

*~~Simily all entities deined within the domain must have name, purpose, responsibilities. Having a name is not enough as names may be the same in different domains repsesenting different roles of the same physical object.~~*

1. *~~The purpose of a SysML model is to provide a specification from which the system can be implemented. A SysML model in a definitive and unambiguous statement of the requirements. It stands to reason that implementers must understand SysML. The stakeholders must also be able to understand the model to a degree. However the requirements in written form can be traced and coverage metric made. Attention to tracing means that stakeholders do not have to verify the model alleviating the need for stakeholders to understand it.~~*
2. *~~Possible need to put this first:- Assumptions on the knowledge that reader must have to understand this article.~~*

*~~Knowledge of SysML~~*

*~~Knowledge of design in real time Systems~~*

*~~Knowledge of control system design~~*

1. *~~Example of AI to dynamically choose a solution depending external conditions.~~*
2. *~~Start with a summary for the article~~*
3. *~~Bad practise to use we all the time~~*
4. *~~Footnote 39 Activity diagrams can be also drawn without swim lanes. Robinson dilemma. The concept of allocation is at odds with the description on the control system. These must be reconciled. See red markup.~~*
5. *~~Example of domain synchronisation is the German English electric adapter plug~~*
6. *~~Example of domains train door HMI plus the door fragment from application domain~~*
7. *~~Example of a continuous system. E.G engine control FADAC~~*
8. *~~Gaps between heading and text sometimes there is not one~~*
9. *~~Mathematical models can be used in continuous systems and the section domain properties~~*
10. *~~You don’t actually explain what a control system is compared to a model !!~~*
11. *~~There seems to be static and dynamic synchronization between domains. Is this really true is it just synchronization with different synchronising methods. No just one synchronisation specification. Only some of them need to be implemented in the real time system.~~*
12. *~~Example on the train door the question is the real object door operator part of the application domain after it is split into proxy and real object? Whats the answer to this.? No this is a different domain. Control domain and hardware counterpart domain.~~*
13. *~~Every domain has a counterpart control domain.? NO~~*
14. *~~The operator object seems to be the one with the most context. For example batch or train operator to control the train and doors. The operator is part ofthe system and has functionallity allocated to It. It needs to trackedand controlled likeany other hardware object~~.*
15. *~~The sequence diagram shows how objects between domains are synchronised. The actor in a sequence diagram belongs to a different domain than the instant time line objects.~~*
16. *~~SysMl model is a perfect model because it represents requirements.~~*
17. *~~Use case alternatives are implemented by different allocations. Check the success condition and minimum guarantees in a use case.~~*
18. *~~Allocation concept is still not too clear. Control system allocation and ordinary allocation !!!~~*
19. *~~Control system what should a sysml model consist of as a bare minimum. The concept of control system muddled the waters!!!! Or does it!!!!!~~*
20. *~~In the jet engine model make sure that all properties actually belong to something i.e. a block.~~*
21. *~~Make the model example to show the domain model and then the derived control system model using jet engine? Not done~~*
22. *~~Models are never wrong but they may express requirements which can be disputed~~*
23. *~~Because a model expresses requirements it is modelled assuming perfect and instantaneous behaviour . The behaviour of the system under fault conditions plays no part in an application domain model because only those requirements pertinent to the domain are modelled.~~*
24. *~~The object with the most context is either constructed or controlled. It represents what the operator interacts with. This object is the proxy to the operator. i.e batch thrust. This comes about only after allocation of functionallity to an operator~~*
25. *~~Allocation in the control system is slightly different than allocation in the model before splitting into proxies and actual hardware. The allocation of responsibilities to the operator and proxy is shown in a sequence diagram. This allocation can all functions to operator. As the functionality is increased more and more of the functionality is allocated to the proxy i.e self drive car. TRUE or NOT. Confused statement, Operators and software appear when functionality is allocated to them~~*
26. *~~Proxies:-~~*
27. *~~Created to present co-ordinated behaviour between physical objects chosen for the solution~~*
28. *~~To represent roles which are relationships which have behaviour and can be created and deleted~~*
29. *~~Actual behaviour of an external physical object e.g. door open and closed.~~*
30. *~~V model for the above physical objects mirrored bythe proxiesfar left and right are objects of most context. Draw a diagram to show this~~*
31. *~~Separate the state machine landing brakes between door reverser and landing brake for point 3 in 45.~~*
32. *~~Final section on architecture to describe the more up to date approach taken by any companies when using SysML. Treat the previous sections as a requirements analysis process. There needs to a synchronisation process between the logical model and final architectural example.~~*
33. *~~Fault scenarios modelled by different allocations~~*
34. *~~How to find physical objects and what physical objects are~~*
35. *~~Example comms system and the OSI stack~~*

*~~Ultimate purpose and recipient receiving the service is often not in view when modelling the control system. Nearly always we have a subsystem which is supporting the ultimate purpose.~~*

1. *~~Including certain objects in a model can be problematic if they are not extracted enough. For example ask the reason why trains have doors. This provides a safe access from carriage to platform. In other words the door does not open if train is not stopped in station with platform and door adjacent.~~*
2. *~~Control system design initially does not have an operator as in pilot. Allocations of control model to live, hard and soft ware is made later.~~*
3. *~~Include an architectural example after allocation to ware’s where does the architectural model come in and what does one look like. Use OSI stack as an example using the two layers datalink and application. Or put this into the example section a telephone system.~~*
4. *~~Not sure about couterparting to only roles. Now think engine and aircraft for example are the physical counterparts for thrust control and flight.~~*
5. *~~Adding 3 more examples car head lights(battery, charging circuit) lift (lift floor) traffic lights (cross section, single lane, lane allocating lane part to lane)~~*
6. *~~allocation of requirements to architectural elements from BorgWarner systems engineering lead job spec~~*
7. *~~re-write of the concept of allocation and abstraction. Allocation of behaviour to abstract objects. Allocation of abstract objects to more concrete objects. The definition of abstraction in terms of requirement, solution and solution set.~~*
8. *~~Modern concept of requirements are the behaviour statements (statements without objects). The modern concept of architecture appears to be any object (abstracted or not) introduced into the model. Allocation assigns behaviour to objects. Both sequence and activity with swimlanes are models showing allocation. A sysml model needs objects to fully explain the requirements of the system. Control system also needs objects.~~*
9. *~~Allocation – dealing with errors and allocating behaviour to architectural objects~~*
10. *~~Allocation – confusion when using in context of a resource and allocating for different implementations.~~*
11. *Make an index*
12. *Application and service domain transporter and robot instruction set*
13. *~~Control system proxy and physical object are in the same domain. Control system proxy is a role of the physical object. The same set of physical objects play different roles in different domains. Illustrate with call model MMI allocation application. Model these three domains.~~*
14. *~~Example of synchronisation in call handling are protocols. This synchronises objects in the same domain. In addition we have mechanisms to synchronise between domains.~~*
15. *~~Difference between implementation allocation and resource allocation not clear~~*
16. *~~The model which is used to derive the control model has relationship dynamics and object dynamics. The control model is derived after the physical objects which provide the solution are decided on. Therefore looks as if call is counterparted to switch, lift is counterpartted to user request. User request and call are the control objects for the physical objects lift and switch.In the control system design they are spit aprt from their real physical objects. Note that resource allocation is a separate domain. Need to re-work models and amend the control system writeup.????????????? Not so simple~~*
17. *~~Consider introducing a software architecture model and tracing to written requirements. This might be required because many projects only have the software architectural model. for the control system.~~*
18. *Creative commons license*

Architecture

design patterns 11

input and output 25

layered 25

middle layer 25

OSI model 11

sensors and actuators 25

SOA 11

Attack radar system 15

Automated landing system 15

C++ 16

assembler 16

compiler 16

procedure call 17

Car headlights 38

Communication system 17

call 18, 32

call handling 32

first in first out’ strategy 34

protocol stack 17

Protocols 32

subscriber 32

telephone system 17

Control system 17

access, train doors 18

architecture 25

call 18

centralised 19

context 24

coordinator 18

delegation 24

devices 25

distributed 19

Entity relation model 19

event driven 26

FADAC 19

hardware 19, 25

hierarchical view 24

human operator 18

interface 25

interface facade 18

liveware 19, 25

object orientated approach 18, 24

operator 25

physical objects 40

proxy 18, 20, 40

role 18, 19, 40

scope 24

Shlaer/Mellor 17

software 19, 25

solution choice 19

system design 19

temperature regulator 18

thrust, jet engine 18

V model 15, 24, 32

Domain 7

aeroplane application 16

analysis 39

application 11, 12, 39

architecture 8, 13

component 14

definition 16

divide and conquer 4, 10

is counter parted by 16

knowledge 12

logical 13, 15

mapping rule 16

MMI service 12

model 13, 15

object 14

physical 13, 15

property 13

purpose 12

responsibilities 12

service 10, 11, 12, 39

subject matter 10, 11, 12

synchronisation 14, 16

wheel retraction system 16

world state 16

FADAC

EPR,N1,N2 20

FADEC 29

bleed 29

multi-dimensional safety envelope 29

throttle 29

thrust 29

functional decomposition 10

Gas light 15

Input/output 17

adapter 17

devices 17

German English electric adapter plug 17

Legacy systems 5

Lift 26, 35

Model

analyst 14

architecture 13

behaviour 13

diagrams 15

hierarchy 5, 9

structure 13

tables 15

Model ship 4

Physical

arrangement 15

deployment 15

flows 15

structural 15

Project manager 5

Requirement 40

abstraction 4, 7, 10, 13, 17

agile 5

ambiguous 5

analysis 5

behavioural 5

coverage 10

derived 10

essential 7, 10, 13

solution set 7, 10, 14, 17

stakeholder 10, 11

statement 5, 10

story boarding 5

system engineering 9

Resource allocation

shared resources 12

Robot 5, 39

configuration facility 15

instruction sets 15, 16

robotics service domain 15

software 15

steady state velocity schedule 26

transporter 15, 16, 17, 28

Science

Einstein 14

maths 14

theories 14

Solution 15

design process 5

end solution 14

implementation 8

operating modes 15, 39

State Machine 22

SysML 3, 40

activity diagram 5, 6, 26

allocation 5, 8, 12, 14, 18

block diagram 24

model 11

Object block 5

parametrics 26

relation 14

sequence diagram 5, 6, 25, 26

state 26

state event matrix 6

state machine 5, 26

structural diagram 14

trace relation 10, 11

use case 5, 6

System

Architect 13

architecture 13

design team 14

project manager 14

Tools

Matlab 4, 25

Simulink 4, 25

Traffic light 15, 17, 36

segment 36

Train door

access 31

lock 31

proxy 31

Transporter 39

robot 4, 11

Water boiler 6

kettle 8

1. SysML can support a broad range of systems and systems of systems however its primary intention is for systems which operate in real time. [↑](#footnote-ref-1)
2. A single person may have many acting roles when they are part of the system.. The system beneficiary is also human by definition. The beneficiary just receives the benefit but does not further process it as part of the system. Be wary of the confusion that can arise. [↑](#footnote-ref-2)
3. The pillars of SysML include Structural, Behavour, Requirements and Parametrics [↑](#footnote-ref-3)
4. SysML Distilled Lenny Delligatti defines the term ROI [↑](#footnote-ref-4)
5. This is discussed in a section below [↑](#footnote-ref-5)
6. For example simulation using simulink [↑](#footnote-ref-6)
7. Known requirements need not be written down but if they are they are usually held in a database such as DOORS, [↑](#footnote-ref-7)
8. For example an engine limiter may be shown in an activity diagram as a swim lane. The reader should be aware that blocks are classified as structural rather than behavioural in SysML. [↑](#footnote-ref-8)
9. Sequence digrams show only how objects are synchronised. [↑](#footnote-ref-9)
10. Equipment failures in the implementation do occur and must be catered for. These requirements are modelled in SysML in the same way as any other requirements. See section on allocations to see one way of dealing with errors. [↑](#footnote-ref-10)
11. SysML models can easily be compiled to environments which can be executed. This is one of the advantages of SysML [↑](#footnote-ref-11)
12. The water batch still exists but does not have behaviour. [↑](#footnote-ref-12)
13. The design process produces derived requirements which causes the model and requirement statements to be hierarcheral [↑](#footnote-ref-13)
14. It is sometimes thought that only the imported requirement statements, for example ones held in DOORS, are the requirements of the system. This is however not correct. Every aspect of a model is also a requirement which the implemented solution must meet. [↑](#footnote-ref-14)
15. There will be many additional requirements discovered by the modelling process. These requirements should be expressed in sentence form and added to the original set. This gives stakeholders without SysML knowledge a way of reviewing the model. These new statements should also be traced. [↑](#footnote-ref-15)
16. The process of splitting a system in its separate subject matters takes time to produce. This can be months or years. It is equivalent to the production of conventional requirement document. [↑](#footnote-ref-16)
17. In a telephone exchange the OSI stack has been developed to separate the concerns of call handling from that of communication protocols dealing with synchronisation. For example HTTP transport layer is treated as a separate subject matter and would be separated out in the model in a lower layer. [↑](#footnote-ref-17)
18. This software develop strategy is also called separation of concerns. [↑](#footnote-ref-18)
19. In practise complete decoupling among modules is hard to achieve and some trade off is often required. [↑](#footnote-ref-19)
20. Provided that the model is properly constructed using a domain architecture. See below. [↑](#footnote-ref-20)
21. A service domain is reusable in another application. An application domain cannot be reused by definition. [↑](#footnote-ref-21)
22. The transporter route was implemented in terms of the robotic instruction set. [↑](#footnote-ref-22)
23. For example wait to be seated sign in a restaurant is an example of such a domain. [↑](#footnote-ref-23)
24. These are the derived requirements which come about by requirement analysis and design. The initial requirement statements held in a Data Base (for example DOORS) are usually imported via a CASE tool and are traced to model elements. [↑](#footnote-ref-24)
25. All domains contain named objects. These objects may be just activity/behavioural fragments. This is supported by SysML. A domain may contain abstract objects. An abstract object may be implemented by different physical objects. [↑](#footnote-ref-25)
26. When trying to divide a complicated confused subject matter into separate domains one should approach this with a degree of pragmatism. Most projects are commercially driven so spending a large amount of time on this type requirement analysis must be justified. At the end of day there will always be a sense of incompletion. Apply the 80% 20% rule to this activity. [↑](#footnote-ref-26)
27. A simple architecture used in many systems is the client – server. Design patterns such as model view controller also describe architectures. [↑](#footnote-ref-27)
28. For example rotor speed may be determined by six or more independent readings and calculations and compared. [↑](#footnote-ref-28)
29. In this case the architectural domain could show the domain hierarchy in a class diagram. Note that the subject matter is completely different from the system purpose. [↑](#footnote-ref-29)
30. Other ways include functional decomposition and other ad hoc methods which are a mixture of functional and subject matter partitioning. [↑](#footnote-ref-30)
31. In SysML mathematical modelling is performed using parametrics to represent the system constraints. [↑](#footnote-ref-31)
32. The reader should be aware that the intent and purpose of a domain that uses physical object names will not necessary be to model the physical world. [↑](#footnote-ref-32)
33. The requirements specified in an application domain must be implemented by service domains is an example of a static relation which does not change through time. [↑](#footnote-ref-33)
34. The allocation of functional requirements to system architectural components is a common exercise in system design [↑](#footnote-ref-34)
35. The SysML allocation is broader than this. The SysML specification does not dictate what it is used for. Rather it suggests uses through examples. [↑](#footnote-ref-35)
36. Note that this does not necessary mean physically inside. [↑](#footnote-ref-36)
37. A model using abstractions of real world objects is the OO approach for system design. The control system can be specified using these objects as proxies. A context diagram can be drawn showing the external physical objects that the control system needs to track and control. Using a physical centralised control system is a design decision. The logical model should avoid making such assumptions. [↑](#footnote-ref-37)
38. An activity diagram with swimlanes best illustrates the concept of allocation. This type of diagram makes it very clear what behaviours are allocated to which physical/logical objects. [↑](#footnote-ref-38)
39. A common mistake in modelling is to imagine all the failure conditions and try to model them. A model should represent only requirements as in the different modes of operation. [↑](#footnote-ref-39)
40. The state of a domain is the sum total of property values used by the domain in the model. In SysML this is often represented by a state diagram with an object property value of state. [↑](#footnote-ref-40)
41. From Shlaer/Mellor [↑](#footnote-ref-41)
42. I.e. A geographical change would not directly change historical events. [↑](#footnote-ref-42)
43. Note that C++ and assembly are separate worlds. A change in the definition of the C++ language will have no impact on the existing definitions of assembler code. [↑](#footnote-ref-43)
44. The sequence diagram uses actors to represent objects in other domains and shows synchronisation using object instance time lines. This SysML activity can take up a large chunk of the development effort. [↑](#footnote-ref-44)
45. For example the World Wide Web. [↑](#footnote-ref-45)
46. For example the compiler is stand alone and an interpreter is part of the run time system. [↑](#footnote-ref-46)
47. Today this methodology is obsolete but the main concepts are still valid. [↑](#footnote-ref-47)
48. Activity diagrams can be also drawn without swim lanes. This is advocated by Robertson as a starting point for requirement analysis. At this point in the requirement specification the model may consist of a simple written description defining what is needed. Or more formally expressed as a set of use cases. [↑](#footnote-ref-48)
49. It’s possible for the solution set to contain only one solution. In this situation one has to ask the question is it worth doing a SysML model? [↑](#footnote-ref-49)
50. The control object and the physical object are really the same object only split apart. [↑](#footnote-ref-50)
51. Physical objects themselves are often modelled in an architectural model, Note this is a separate domain. [↑](#footnote-ref-51)
52. If the physical object only has one role then the control object represents the behaviour of the physical object itself [↑](#footnote-ref-52)
53. Note the operator is not the ultimate human beneficiary but is part of implementation of the system. [↑](#footnote-ref-53)
54. The physical object may not be tangible as in a co-ordinating object. [↑](#footnote-ref-54)
55. The physical object may have only one role in which case the behaviour of the object and role are the same. [↑](#footnote-ref-55)
56. Peopleware means actual people [↑](#footnote-ref-56)
57. Control system objects can also be implemented fully in hardware as in a centrifugal governor on a steam engine. [↑](#footnote-ref-57)
58. Note an equivalent activity diagram could also be drawn. [↑](#footnote-ref-58)
59. The context diagram is not a formal SysML diagram. It can be drawn using internal block and block diagrams. [↑](#footnote-ref-59)
60. The application and service domains may also be represented in a similar diagram. [↑](#footnote-ref-60)
61. The proxies form a mirror image of the ordered physical objects. Objects closer to the mirror boundary have less context and knowledge of the system than objects further away. [↑](#footnote-ref-61)
62. A control system designed with delegation, following a V formation, uses a layered architectural. [↑](#footnote-ref-62)
63. This type of architecture meets the needs of the application domain. Other service domains may need to have a different architecture to explain how they can be constructed. [↑](#footnote-ref-63)
64. A consequence of this is that models take no time to execute and operates perfectly if you prefer to view a sequence diagram as an ordered sequence. For example the event engine started instantly changes the the state of the flight control proxy and pilot interface. (There is no time order) [↑](#footnote-ref-64)
65. Note that robot and transporter are in different domains [↑](#footnote-ref-65)
66. Such as rotor speed and temperature [↑](#footnote-ref-66)
67. Mathematical equations would be placed in the model as constraints. The formulas themselves are constructed outside the model and tested using tools such as Matlab. [↑](#footnote-ref-67)
68. Note that the beneficiary is the person receiving this service. This is access from the train to the platform. The beneficiary and the door operator may be the same person. [↑](#footnote-ref-68)
69. An example would be ‘please wait to be seated’ in a restaurant. [↑](#footnote-ref-69)
70. The conflict resolution would stop two lifts moving to the same floor simultaneously. [↑](#footnote-ref-70)
71. The traffic light allows the drivers of cars to interface to the traffic light system. It therefore belongs to the MMI service domain. [↑](#footnote-ref-71)
72. This is implementation allocation using the relation ‘is allocated to’ [↑](#footnote-ref-72)
73. Operating modes are the systems supported allocations and can be shown in an allocation diagram with conditions of deployment. [↑](#footnote-ref-73)