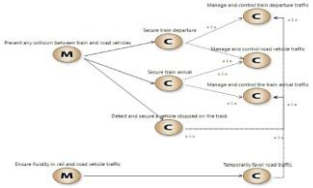
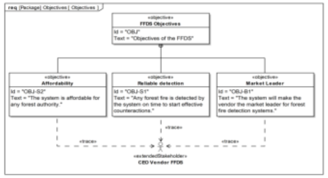
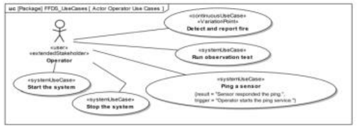
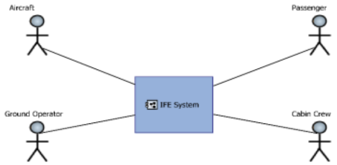
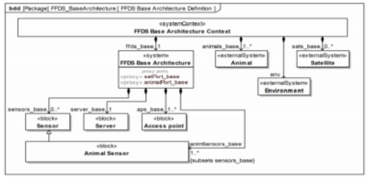
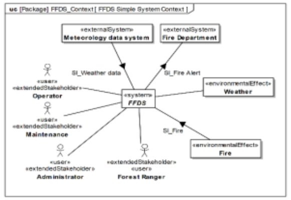
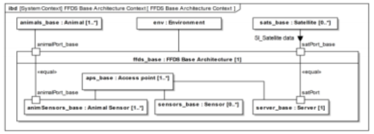
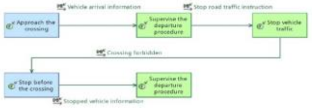
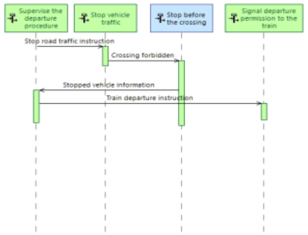
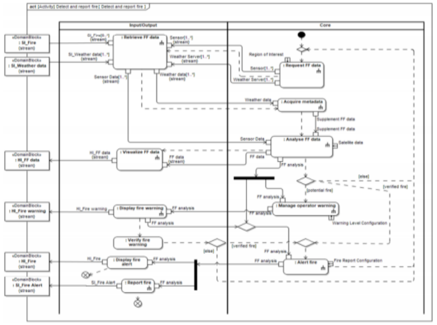
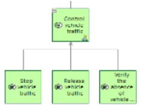
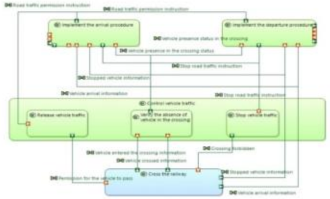
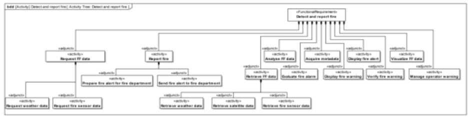
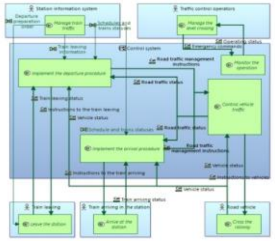
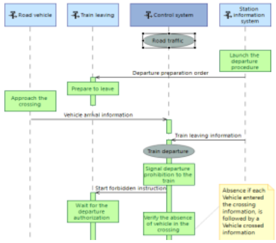
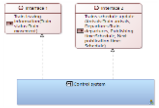
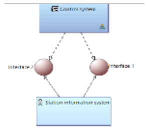
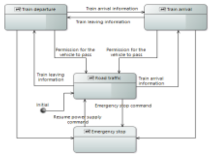
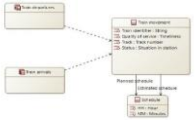
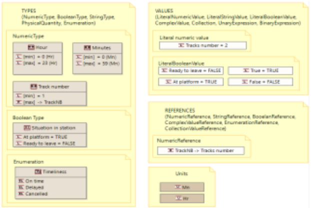
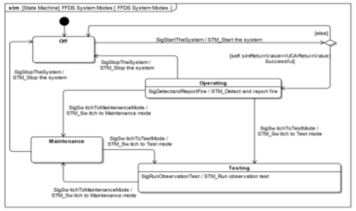
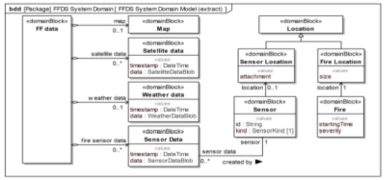
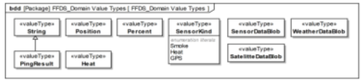
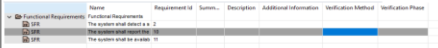
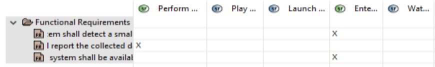
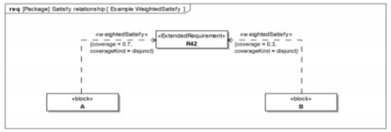
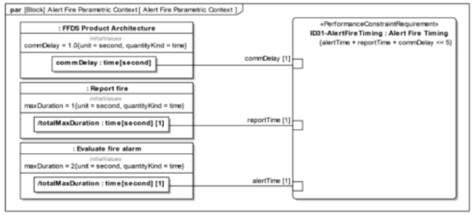
Arcadia	Description	Diagrams	SYSMOD	Description	Diagrams
Transition From Operational Activities	This step enables the transition of Operational Activities to System Functions. It also provides tracebility matrices mapping the Operational Activities to System Functions and the automated updating in case of changes.	For matrices example see *Requirements* below		As SYSMOD does not seperate between different levels of analysis, a transition is not explicitely included in the method. Yet SYSMOD using SysML enables traceability and traceability matrices between elements. Functions for automating transitions and generating traceability matrices are depending on the modeling tool in use and are not considered any further.	For matrices example see *Requirements* below
Define Actors, Missions and Capabilities	This step focuses on defining the actors of the system (who interacts with the system especially via interfaces) and their goals. Main concepts: - Missions - Capabilities - Actors	[MCB] System Missions and/ or Capability Diagram  Manage and control train departure traffic Secure train departure Prevent any collision between train and road vehicles M C Manage and control road vehicle traffic Secure train arrival Manage and control the train arrival traffic Detect and secure a vehicle stopped on the track Ensure fluidity in rail and road vehicle traffic Temporarily favor road traffic It allows to model System Missions and Capabilities and their relationships as well as their connections to the system actors. It is similar to the SysML Use Case Diagram, including the include, extend and generalization relationship.	Describe the System Objectives Identify Stakeholders Identify and Model the System Use Cases and System Processes	SYSMOD also captures high-level goals and needs. It adds additional elements for describing stakeholders (that don't necessarily have to interact with the system but have other interests in the system). Main Concepts: - System Objectives - System Use and Continuous Use Cases - System Processes - Stakeholders => Details Use Case concept and adds System Processes => SYSMOD describes actors and their goals and adds stakeholders and their objectives	[req] System Objectives (+trace relationship to Stakeholders)  req [Package] Objectives [Objectives] «objective» FFDS Objectives Id = "OBJ" Text = "Objectives of the FFDS" «objective» Affordability Id = "OBJ-S2" Text = "The system is affordable for any forest authority." «objective» Reliable detection Id = "OBJ-S1" Text = "Any forest fire is detected by the system on time to start effective counteractions." «objective» Market Leader Id = "OBJ-B1" Text = "The system will make the vendor the market leader for forest fire detection systems." «trace» «extendedStakeholder» CEO Vendor FFDS [uc] System Use Cases (one per actor)  uc [Package] FFDS_UseCases [Actor Operator Use Cases] «continuousUseCase» «VariationPoint» Detect and report fire «user» «extendedStakeholder» Operator «systemUseCase» Run observation test «systemUseCase» Start the system «systemUseCase» Stop the system «systemUseCase» Ping a sensor {result = "Sensor responded the ping.", trigger = "Operator starts the ping service."}
	Main Concepts: - Actors - System	[CSA] Contextual System Actors  Aircraft Passenger IFE System Ground Operator Cabin Crew It is used to have a high-level overview of the system and the system actors.	Analyse the Problem (Is primarily done outside the model. But the results are captured in the model) Describe the System Idea Identify System Context Describe Base Architecture	SYSMOD adds additional and more specific information and model elements to describe how the system is embedded in its environment and what are the technical and architecture decisions preset at project start. Main concepts: - System with properties SystemIdea and ProblemStatement - Base Architecture - System Context with Actors (User, External System and Environmental Effects) => SYSMOD provides a wide variety of structural specifications in the analysis process => But freedom to choose how detailed the structural description can get (For base archtiecture: from beermat to [bdd]. For system context: from [bdd] and [uc] high-level system context to [ibd] detailed system context)	[bdd] System with Problem Statement and System Idea [bdd] and [ibd] for Base Architecture and System Context  bdd [Package] FFDS_BaseArchitecture [FFDS Base Architecture Definition] «systemContext» FFDS Base Architecture Context ffds_base 1 animals_base 1..* sats_base 0..* «system» FFDS Base Architecture proxy ports «proxy» satPort_base «proxy» animalPort_base «externalSystem» Animal «externalSystem» Satellite env «externalSystem» Environment sensors_base 0..* server_base 1 aps_base 1..* «block» Sensor «block» Server «block» Access point «block» Animal Sensor animSensors_base 1..* {subsets sensors_base}  uc [Package] FFDS_Context [FFDS Simple System Context] «externalSystem» Meteorology data system «externalSystem» Fire Department SI_Weather data SI_Fire Alert «user» «extendedStakeholder» Operator «system» FFDS «environmentalEffect» Weather «user» «extendedStakeholder» Maintenance SI_Fire «environmentalEffect» Fire «user» «extendedStakeholder» Administrator «extendedStakeholder» «user» Forest Ranger  ibd [System Context] FFDS Base Architecture Context [FFDS Base Architecture Context] animals_base : Animal [1..*] env : Environment sats_base : Satellite [0..*] SI_Satellite data animalPort_base satPort_base ffds_base : FFDS Base Architecture [1] «equal» aps_base : Access point [1..*] satPort animSensors_base : Animal Sensor [1..*] sensors_base : Sensor [0..*] server_base : Server [1]

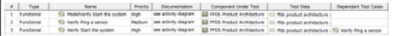
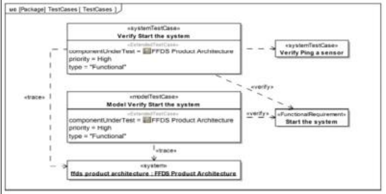
Refine System Functions, describe functional exchanges	In this step the focus is on refining the system functions, by - enriching and detailing the functional breakdown by adding new system functions - and describing data flows and functional chains. Main concepts: - Functions - Functional Exchanges - Function Ports - Data Flow (and Control Flow)	[SFCD] System Functional Chain  Vehicle arrival information Stop road traffic instruction Approach the crossing Supervise the departure procedure Stop vehicle traffic Crossing forbidden Stop before the crossing Supervise the departure procedure Stopped vehicle information Decribes a path/ ordered set of functions and functional exchanges that link them. [FS] Functional Scenarios  Supervise the departure procedure Stop vehicle traffic Stop before the crossing Signal departure permission to the train Stop road traffic instruction Crossing forbidden Stopped vehicle information Train departure instruction Set of functions and the exchanges that link them on a temproal axis.	Model Use Case Activites	In *Model Use Case Activites* the functional decomposition of the System Use Cases is specified. This includes defining the sequence of execution and the object flow between the system functions (activities and actions). Main Concepts: - Activity, CallBehaviorAction, Action - Activity Parameter Node, Object Node, Pin - Object and Control Flow => Modelling scenarios: SysML does not allow to use activities and actions in sequence diagrams. The temporal sequence of execution can be modelled by using the control flow	[act] Activity Diagram (one per System Use Case)  The Activity Diagram uses similar concepts like the Arcadia [SFCD], [SFBD], [SDFB] & [SAB] (for [SAB], see Allocate System Functions to System Actors). It sets functions into an order of execution, allows a breakdown into activities, call behavior actions and actions and the allocation to structural elements.
		[SFBD] System Functional Breakdown  Control vehicle traffic Stop vehicle traffic Release vehicle traffic Verify the absence of vehicle ... Describes a functional hierarchy. Subfunctions can be grouped into a mother function. It is not a strong structural decomposition that forms a synthetic representation. Only the leaf functions (without subfunction) carry the functional description. [SDFB] System Data Flow  Dataflow describes functional dependencies between functions (funtional exchanges connected to function ports).			[bdd] Activity Tree (Use Case Activity Breakdown)  Activity Diagrams do not allow to model the whole functional hierarchy in one diagram due to its encapsulation mechanism. Using the [bdd] allows modelling a tree that depicts a call hierarchy of functions (a function is a sub-function of another function if that function calls it. It is an ownership of execution).

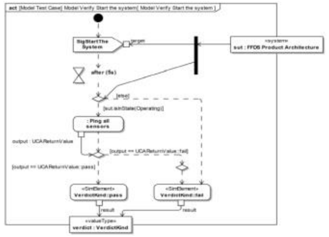
Allocate System Functions to System and Actors	In this step the functions are allocated to the system and actors. It is possible to deduce component exchanges that implement functional exchanges and to model scenarios describing functional exchanges between the system and actors. Main Concepts: - System - Actor - Functions - Functional and Behavioral Exchanges, Physical Links	[SAB] System Architecture  Station information system Traffic control operators Departure preparation order Manage train traffic Schedules and trains statuses Manage the level crossing Operating status Emergency commands Train leaving information Control system Road traffic management instructions Monitor the operation Implement the departure procedure Road traffic status Train leaving status Instructions to the train leaving Vehicle status Control vehicle traffic Schedule and trains statuses Road traffic status Implement the arrival procedure Road traffic management instructions Vehicle status Instructions to the train arriving Instructions to vehicles Vehicle status Train arriving status Train leaving Train arriving in the station Road vehicle Leave the station Arrive at the station Cross the railway [ES] Exchange Scenario  Road vehicle Train leaving Control system Station information system Road traffic Launch the departure procedure Departure preparation order Prepare to leave Approach the crossing Vehicle arrival information Train leaving information Train departure Signal departure prohibition to the train Start forbidden instruction Wait for the departure authorization Verify the absence of vehicle in the crossing Absence if each Vehicle entered the crossing information, is followed by a Vehicle crossed information			[act] Activity Diagram (see above) [seq] Scenarios It is possible to use Sequence Diagrams to model exchanges between the system and the system actors. But in SYSMOD Sequence Diagrams are primarily used in the Architecture Process (see *Revise an Architecture with Scenarios*). [act] Use Case Activities Additionally SYSMOD uses Activity Diagrams to describe exchanges between the system and the actors. => In difference to Arcadias Exchange Scenario SYSMOD specifys the input and output between the system and the system context but not the detailed interaction between the system and the actors in form of functions or activities. (see diagramm at *Model Use Case Activites*)
Define Interfaces and describe Interface Scenarios	In this step the interfaces of the system and the actors are detailed. Additional scenarios are described to specify the dynamic behavior of the system. Main Concepts: - System and Actor - Interface, Port, Exchange Item	[CDI] Contextual Detailed Interface  [CEI] Contextual External Interface  Control system Interface 2 Interface 1 Station information system [IS] Interface Scenario (see other scenario diagrams)	Identify the System Context	SYSMOD using SysML uses the [bdd], [uc] or [ibd] to define interfaces. For example in the System Context or Base Architecture (see above) interfaces can be identified and then be defined by different ports (full and proxy), the SysML InterfaceBlock or the SYSMOD stereotype *<<userInterface>>* . (See also Interface and data model concepts) Main concepts: - System Context, Base Architecture - Ports, Item flow, ValueProperties InterfaceBlock, *<<userInterface>>*	see System Context and Base Architecture

Transverse Modeling	In this step the state machine, the domain elements and the actually exchanged data between components is specified. The state machine can capture modes and states of the system, components, actors and classes (data). The domain knowledge describes high-level semantics related to the domain. Main Concepts: - Modes & States - Classes - Interfaces, Exchange Items, - Types & Values	[MSM] Mode State Machine  Train departure Train arrival information Train leaving information Train arrival Permission for the vehicle to pass Permission for the vehicle to pass Train leaving information Road traffic Train arrival information Initial Emergency stop command Resume power supply command Emergency stop [CBD] Class Diagram Blank - Domain Elements  Train movement Train identifier : String Quality of service : Timeliness Track : Track number [CBD] Class Diagram Blank - Domain Elements  TYPES NumericType Boolean Type Enumeration Timeliness On time Delayed Cancelled VALUES REFERENCES NumericReference Units	Specify System States Model the Domain Knowledge	In SYSMOD state machines are primarily used in the Architecture Process for describing the states of the system or the systems parts and the transition of states. But state machines can also be used in the Analysis Process, e.g. to refine requirements. The domain knowledge defines terms of the domain from the perspective of the system. The domain knowledge is captured in domain objects and can be differentiated into context objects (exchange between system and actors) and system objects (only used inside the system). In SYSMOD the domain knowledge is also called concept or data model. Main Concepts: - States - Block stereotype *<<DomainBlock>>* - Value Types	[stm] State Machine  stm [State Machine] FFDS System Modes [FFDS System Modes] Off [else] SigStartTheSystem / STM_Start the system [self.sinReturnValue==UCAReturnValue::Successful] SigStopTheSystem / STM_Stop the system SigStopTheSystem / STM_Stop the system Operating SigDetectandReportFire / STM_Detect and report fire SigSwitchToMaintenanceMode / STM_Switch to Maintenance mode SigSwitchToTestMode / STM_Switch to Test mode Maintenance SigSwitchToTestMode / STM_Switch to Test mode Testing SigRunObservationTest / STM_Run observation test SigSwitchToMaintenanceMode / STM_Switch to Maintenance mode [bdd] Domain Knowledge  bdd [Package] FFDS System Domain [FFDS System Domain Model (extract)] «domainBlock» FF data map 0..1 «domainBlock» Map satellite data 0..* «domainBlock» Satellite data timestamp : DateTime data : SatelliteDataBlob weather data 0..1 «domainBlock» Weather data timestamp : DateTime data : WeatherDataBlob fire sensor data 0..* «domainBlock» Sensor Data timestamp : DateTime data : SensorDataBlob sensor data 0..* created by «domainBlock» Location «domainBlock» Sensor Location attachment location 0..1 «domainBlock» Fire Location size location 1 «domainBlock» Sensor id : String kind : SensorKind [1] sensor 1 «domainBlock» Fire startingTime severity [bdd] Domain Value Types  bdd [Package] FFDS_Domain Value Types [FFDS_Domain Value Types] «valueType» String «valueType» Position «valueType» Percent «valueType» SensorKind Smoke Heat GPS «valueType» SensorDataBlob «valueType» WeatherDataBlob «valueType» PingResult «valueType» Heat «valueType» SatelliteDataBlob
Additional steps not explicitely mentioned in the Capella System Analysis Activity Browser					
Requirements	Arcadia recommends to use the model itself for expressing requirements (and also feared events) as most of the needs can be fomalized by a model (for example functions or scenarios as functional requirements or specific elements of functional chains for non-functional requirements). In cases where textual requirements are needed Capella provides different types of requirements, tables and matrices. Requirements can also be grouped into requirement packages. Linking between requirements and other model elements is also possible by using the requirements viewpoint. Main Concepts - Tables and matrices - Requirements: System Functional (Interface), System Non Functional (Interface), User	Requirements Table  Name | Requirement Id | Summ... | Description | Additional Information | Verification Method | Verification Phase Functional Requirements | Functional Requirements SFR | The system shall detect a s | 2 SFR | The system shall report the | 10 SFR | The system shall be availab | 11 Requirements - Functions Matrix  | Perform ... | Play ... | Launch ... | Ente... | Wat... Functional Requirements em shall detect a smal | | | | X | l report the collected d | X | | | | system shall be availal | | | | X |	Model Requirements and Risks	SYSMOD recommends to use tables and matrices to capture requirements in textual form. But it also provides the *<<extendedRequirement>>* stereotype that can be applied to any model element to represent a requirement. The *<<risk>>* stereotype can be used for modelling risks. For specific performance requirements in the parametric-context *<<PerformanceConstraintRequirement>>* can be used. Main Concepts: - Tables & matrices - Requirements Diagram - Stereotype *<<extendedRequirement>>* with non-functional requirements (e.g Usability, Business, Reliability, etc.) and the functional requirement - Stereotype *<<risk>>* - SysML and SYSMOD-specific relationships: (Weighted)Verify, (Weighted)Satisfy, etc. - Parametric Diagram + Stereotype *<<PerformanceConstraintRequirement>>*	Requirements Table [req] Requirements diagram  req [Package] Satisfy relationship [Example WeightedSatisfy] «weightedSatisfy» {coverage = 0.7, coverageKind = disjunct} «ExtendedRequirement» R42 «weightedSatisfy» {coverage = 0.3, coverageKind = disjunct} «block» A «block» B [par] Parametric diagram  par [Block] Alert Fire Parametric Context [Alert Fire Parametric Context] : FFDS Product Architecture initialValues commDelay = 1.0{unit = second, quantityKind = time} commDelay : time[second] : Report fire initialValues maxDuration = 1{unit = second, quantityKind = time} /totalMaxDuration : time[second] [1] : Evaluate fire alarm initialValues maxDuration = 2{unit = second, quantityKind = time} /totalMaxDuration : time[second] [1] «PerformanceConstraintRequirement» ID31-AlertFireTiming : Alert Fire Timing {alertTime + reportTime + commDelay <= 5} commDelay [1] reportTime [1] alertTime [1]

<table>
<tr>
<td>Test Cases</td>
<td>Unfortunately I couldn't find sufficient materials and resources to give a detailed view on how Test Cases are used in Capella. This is why I only give a brief introduction. Arcadia uses Test Case for IVV (Integration, Verification & Validation).
A test case is built from scenarios or functional chains to verify a given system integration. They can be grouped into a test campaign.

Main Concept:
- Test Case
- Test Campaign</td>
<td></td>
<td>Specify Test Cases</td>
<td>SYSMOD also uses the concept of Test Cases. Test Cases specify how to verify and validate that the system satisfies the requirements. SYSMOD adds the stereotypes <i>&lt;&lt;ExtendedTestCase&gt;&gt;</i> (additional properties), <i>&lt;&lt;systemTestCase&gt;&gt;</i> (testing the real physical system) &amp; <i>&lt;&lt;modelTestCase&gt;&gt;</i> (testing the model) to the SysML Test Case.

Main Concepts:
- Test Case + SYSMOD stereotypes
- Tables & matrices
- Use Case and Activity Diagrams</td>
<td>[table]

<table>
<tr><th>#</th><th>Type</th><th>Name</th><th>Priority</th><th>Documentation</th><th>Component Under Test</th><th>Test Data</th><th>Dependant Test Cases</th></tr>
<tr><td>1</td><td>Functional</td><td>ModelVerify Start the system</td><td>High</td><td>see activity diagram</td><td>FFDS Product Architecture</td><td>ffds product architecture :</td><td></td></tr>
<tr><td>2</td><td>Functional</td><td>Verify Ping a sensor</td><td>Medium</td><td>see activity diagram</td><td>FFDS Product Architecture</td><td>ffds product architecture :</td><td></td></tr>
<tr><td>3</td><td>Functional</td><td>Verify Start the system</td><td>High</td><td>see activity diagram</td><td>FFDS Product Architecture</td><td>ffds product architecture :</td><td>Verify Ping a sensor</td></tr>
</table>
[uc]


[act]

act [Model Test Case] Model Verify Start the system[Model Verify Start the system]
SigStartThe System
target
«system»
sut : FFDS Product Architecture
after (5s)
[else]
[sut.isInState(Operating)]
: Ping all sensors
output : UCAReturnValue
[output == UCAReturnValue::fail]
[output == UCAReturnValue::pass]
«SimElement»
VerdictKind::pass
«SimElement»
VerdictKind::fail
result
result
«valueType»
verdict : VerdictKind

</td>
</tr>
</table>