



Grant ANR-12-INSE-0011

GEMOC
ANR Project, Program INS

WP5 – GEMOC EXPERIMENTATIONS

**D 5.1.1 Technical Requirements, uses-cases
specification and metrics for experimentations**

Task 5.1

Version 1.0

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

DOCUMENT CONTROL

	- :	A :	B :	C :	D :
Written by Signature	Jérôme Le Noir (TRT)				
Approved by Signature					

Revision index	Modifications
A	
B	
C	
D	

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

Authors

Author	Partner	Role
Ali Koudri	TRT	Lead author
Jérôme Le Noir	TRT	Contributor
Benoît Combemale	INRIA	Contributor
Marc Pantel	IRIT	Contributor
Mélanie Bats	Obeo	Contributor
Cédric Brun	Obeo	Contributor

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

Table of content

1.	Introduction	5
1.1	Purpose	5
1.2	Perimeter	Erreur ! Signet non défini.
1.3	Definitions & acronyms	5
1.4	Intended Audience	5
1.5	References	6
1.6	Summary	6
2.	Uses-cases specification	7
2.1	INRIA: The fUML Case Study	7
2.1.1	The fUML Abstract Syntax	7
2.1.2	The fUML Behavioral Semantics	8
2.2	IRIT use case: Coordination of different semantic variants for the same DSML	9
2.2.1	Application Overview	9
2.2.2	Languages used for these experiments	11
2.2.3	Experimentation	14
2.3	THALES: Functional Radar simulation	14
2.3.1	Application Overview	14
2.3.2	ARCADIA (ARCHitecture Analysis & Design Integrated Approach)	16
2.3.3	Abstract Syntax & Concrete syntax	18
2.3.4	Behavioral Semantics	20
2.3.5	Experimentation	20
3.	Technical requirements	22
3.1	Needs	Erreur ! Signet non défini.
3.1.1	Language Definition needs (M2 Level)	Erreur ! Signet non défini.
3.1.2	Architecture simulation Definition needs (M1 Level)	Erreur ! Signet non défini.
4.	Metrics for Evaluation	23

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

D 5.1.1 Technical Requirements, uses-cases specification and metrics for experimentations

1. Introduction

The GEMOC project targets a language design studio providing methods and tools to ease the design and integration of new MoCCs and executable DSMLs (xDSMLs) relying on the use of proven technologies developed in previous research projects such as Cometa, CCSL, Kermeta and the metamodeling pattern to build xDSML in order to define:

- Modeling languages with associated methods and tools for the modeling of both MoCCs and xDSMLs;
- A single cooperative heterogeneous execution framework parameterized by the MoCCs and xDSMLs definitions;
- A global MoCCs and xDSMLs design methodology encompassing these two items;

A formal specification of the previous cooperation framework to prove its completeness, consistency and correctness with respect to the cooperative heterogeneous model execution needed.

1.1 Purpose

This document aims at defining the GEMOC application use cases, related stakeholders and concerns.

Three case studies will be implemented for the validation of the solutions established in GEMOC. The industrial case studies rely on several heterogeneous cooperative xDSMLs relying on various MoCCs that will be implemented using the GEMOC Studio provided by WP4, using the processes, methods and tools defined in WP1 and WP2 and the formal framework provided by WP3. The experiments will first define the languages, then the models and validate the proposed technologies through the heterogeneous simulation of models.

This document defines the context in which the GEMOC WP1, WP2, WP3, WP4 prototyping results will be evaluated. It provides the description of the expected capabilities of the GEMOC studio and explaining how the different tools will be experimented, the business goals that will be targeted and the criteria that will be retained to evaluate the quality of the results.

1.2 Definitions & acronyms

- **MoCC: Model of Concurrency and Communication**
- **AS: Abstract Syntax**
- **DSA: Domain Specific Action**
- **DSE: Domain Specific Event**
- **DSML: Domain Specific Modeling language**
- xDSML: executable DSML
- CCSL: Clock Constraints Specification Language
- SAM: Structured Analysis Model
- AADL: Architecture Analysis and Design Language
- SAE: Society of Automotive Engineers
- ARCADIA: ARChitecture Analysis & Design Integrated Approach

1.3 Intended Audience

This document mainly targets GEMOC in WP1, WP2, WP3 and WP4 partners. However, it may also be of interest to those involved in the field of architecture engineering at Thales.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

1.4 References

- GEMOC project B document
- GEMOC D4.1 “Architecture document”
- fUML metamodel
- Object Constraint Language (OCL)
- Semantics of inter-object communications mechanisms"
- [Gamma:1995:DPE]
- ISO/IEC 42010 Systems and Software Engineering - Architecture Description

1.5 Summary

This document defines the context in which the GEMOC studio prototyping results will be evaluated. It provides the description of the expected capabilities of the GEMOC studio and explaining how the different tools will be experimented, the business goals that will be targeted and the criteria that will be retained to evaluate the quality of the results.

Several aspects must be considered in GEMOC related to the coordination of heterogeneous Domain Specific Modeling Languages for the Validation & Verification of complex systems. The three use case providers will target different aspects.

Section 1 defines the scope of the document and presents the document structure.

Section 2 describes an introduction of the industrial and academic use cases for the GEMOC studio experimentation.

Section 3 identifies requirements, which have to be supported by the GEMOC studio.

Finally, section 4 provides the metrics definition, which will be used for the evaluation.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

2. Uses-cases specification

Several aspects must be considered in GEMOC related to the coordination of heterogeneous Domain Specific Modeling Languages for the Validation & Verification of complex systems. The three use case providers will target different aspects:

- First, provide a real size common example for the various technologies provided by the GEMOC project, and especially the specification using Abstract Syntax(AS), Domain Specific Action(DSA), Domain Specific Event(DSE) and Model of concurrency and communication (MoCC);
- Then, consider several DSMLs from the same family with different semantics variants expressed at the DSA, DSE and MoCC levels relying on the same AS;
- Last, experiment the coordination of different DSMLs relying on different MoCCs using the composition operators.

2.1 INRIA: The fUML Case Study

This first use case will allow early experiments with the various technologies provided by GEMOC. It is based on the fUML subset of UML.

The *Semantics of a Foundational Subset for Executable UML Models* (fUML) is an executable subset of UML that can be used to define the structural and behavioral semantics of systems. It is computationally complete by specifying a full behavioral semantics for activity diagrams. This means that this DSL is well-defined and enables implementers to execute well-formed fUML models (*here execute means to actually run a program*).

As an example, Figure 1 shows an executable fUML model representing the activity of our team when we meet for work sessions. We are used to first having a coffee while talking together about the latest news. When we finish drinking our coffee and talking, we begin to work.

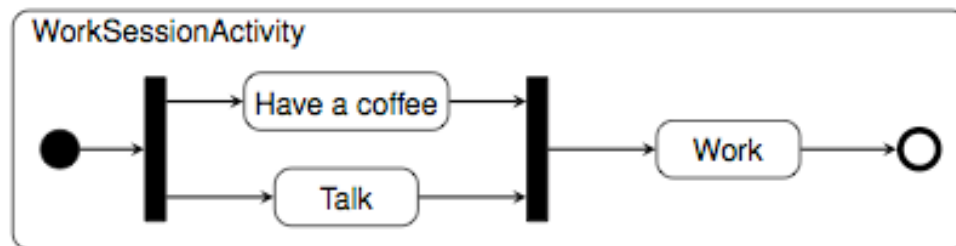


Figure 1 : Activity of the members of our team during our work sessions

The fUML specification includes both a subset of the abstract syntax of UML, and an execution model of that subset supported by a behavioral semantics. We introduce these two parts of the specification in the rest of this section.

2.1.1 The fUML Abstract Syntax

Figure 2 shows an excerpt of the fUML metamodel corresponding to the main concepts of the abstract syntax.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

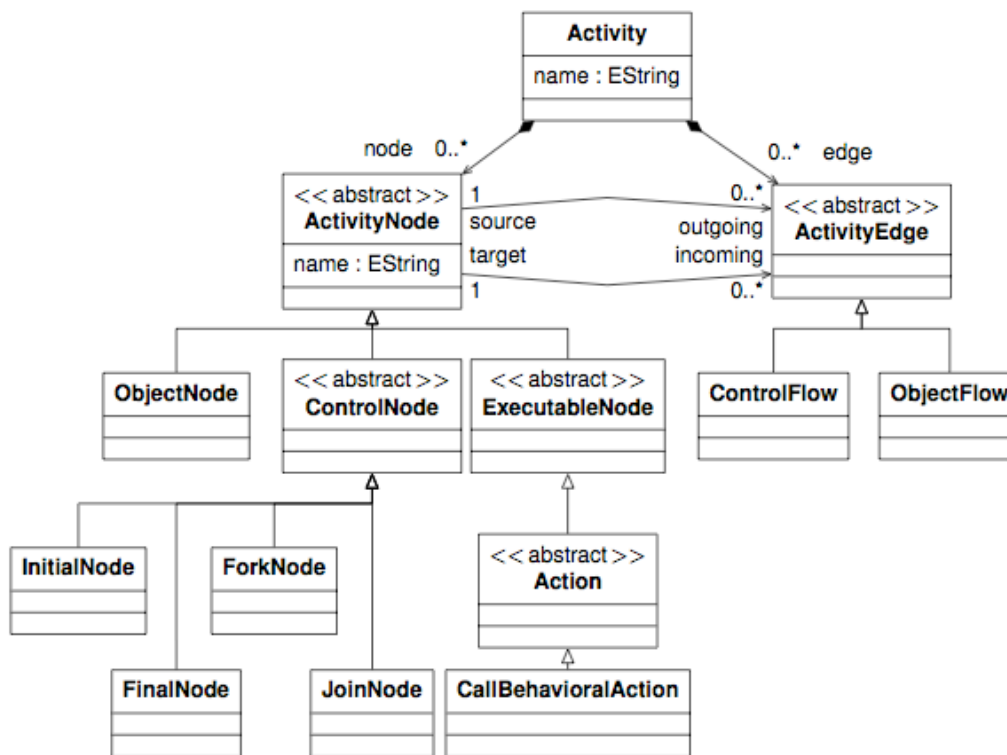


Figure 2 : Excerpt of the fUML Metamodel

The core concept of fUML is *Activity* that defines a particular behavior. An *Activity* is composed of different elements called *Activity Nodes* linked by *Activity Edges*. The main nodes that represent the executable units are the *Executable Nodes*. For instance, *Actions* are associated to a specific executable semantics. Other elements define the activity execution flow, which can be either a control flow (*Control Nodes* linked by *Control Flow*) or a data flow (*Object Nodes* linked by *Object Flow*).

The example in Figure 1 uses an illustrative set of elements of the abstract syntax of fUML. The start of the *Activity* is modeled using an *Initial Node*. A *Fork Node* splits the control flow in two parallel branches: one for the *Action* of having a coffee, the other for the *Action* of talking to each other. Then a *Join Node* connects the two parallel branches to the *Action* of working.

Of course, the abstract syntax also includes additional constraints in the metamodel to precise the well-formedness rules (a.k.a. static semantics). For example, such an additional constraint expresses that control nodes can only be linked by control flows. fUML uses the Object Constraint Language (OCL) in order to define those constraints.

We refer the reader to the specification of fUML for all the details about the comparison with UML2 and the whole description of the fUML metamodel.

2.1.2 The fUML Behavioral Semantics

To support the execution of models, fUML introduces a dedicated *Execution Model*. The activity execution model has a structure largely parallel to the abstract syntax using the *Visitor* design pattern [Gamma:1995:DPE] (called *SemanticVisitor*). Note that although the semantics is explained using visitors, which are rather at the implementation level, it is left open by the fUML specification to implement the language using other means than

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

visitors.

In addition, to capture the behavioral semantics, the interpretation needs to define how the execution of the activity proceeds over time. Thus, concepts are introduced in the execution model for which there is no explicit syntax. Such concepts support the behavior of an activity in terms of *tokens* that may be held by nodes and *offers* made between nodes for the movement of these tokens.

Based on the execution model, the specification denotationally describes the behavioral semantics of fUML using axioms in first order logic. Moreover, a reference implementation of the fUML semantics has been proposed in Java (Cf. <http://portal.modeldriven.org>). Both define the *domain-specific actions* (i.e., the behavioral semantics of the domain-specific concepts defined in the abstract syntax) as a concrete implementation of the visitor, including a deeply scattered scheduling of such domain-specific actions. We refer to the latter concern as (part of) the *Model of Computation* (MoC) of the language. Such an implementation prevents the reuse of a MoC for different DSLs (e.g., the fact that all the domain-specific actions should run in sequence is a behavioral specification that can be reused in many domains), as well as its easy replacement with another one for the same DSL. Indeed, several semantic variation points exist in the MoC. As stated by the specification itself, some semantic areas “are not explicitly constrained by the execution model: The semantics of time, the semantics of concurrency, and the semantics of inter-object communications mechanisms” [X].

2.2 IRIT use case: Coordination of different semantic variants for the same DSML

The main purpose of this use case is to experiment the specification of various Domain Specific Modeling languages (DSMLs) from the same family with different semantics variants. The chosen DSMLs are very common in the development of safety critical systems. Model execution is used both to validate that the model is a correct rendering of the designer purpose and to verify that a design model is a refinement of the corresponding specification model.

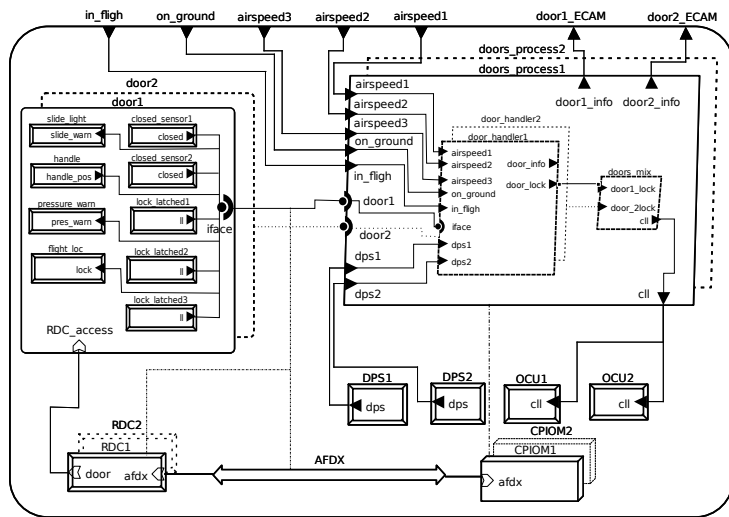
2.2.1 Application Overview

This use case relies on different simple yet realistic subsystems extracted from previous use cases developed in other projects. The real work inside GEMOC is not to design new models but to implement the various xDSMLs in order to be able to Validate & Verify the corresponding previously existing models.

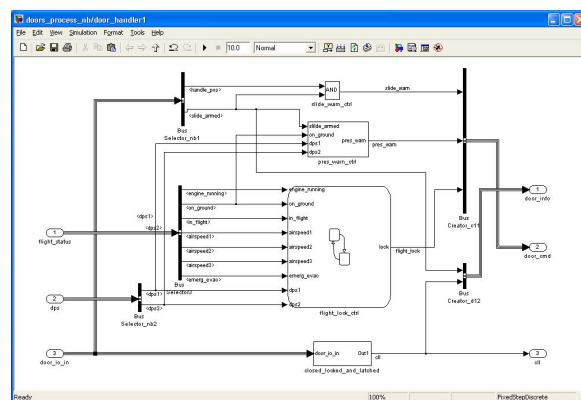
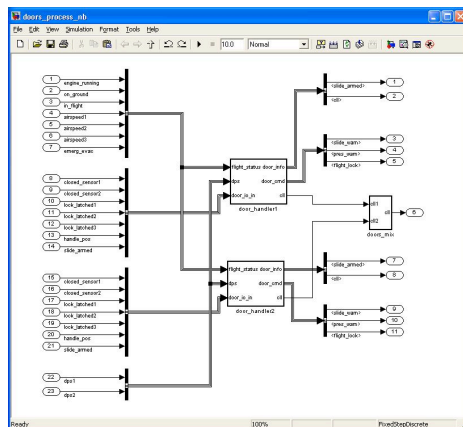
2.2.1.1 Door Management System

The models of the Door Management System from the Airbus A350 were developed in the CESAR project using the AADL/Simulink/Stateflow DSMLs (mainly components and finite automata). The validation will assess that the doors of the plane cannot be opened when the cabin pressure is too low. This model mainly target redundancy issues. It contains an error that appears only when the plane has more than three doors. A first executable model of the whole system is provided in Simulink/Stateflow. Then, the architecture of the system is designed using AADL and the behaviour of each part is designed using Simulink/Stateflow. The verification aims at assessing that both models behave in the same way.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	



AADL models

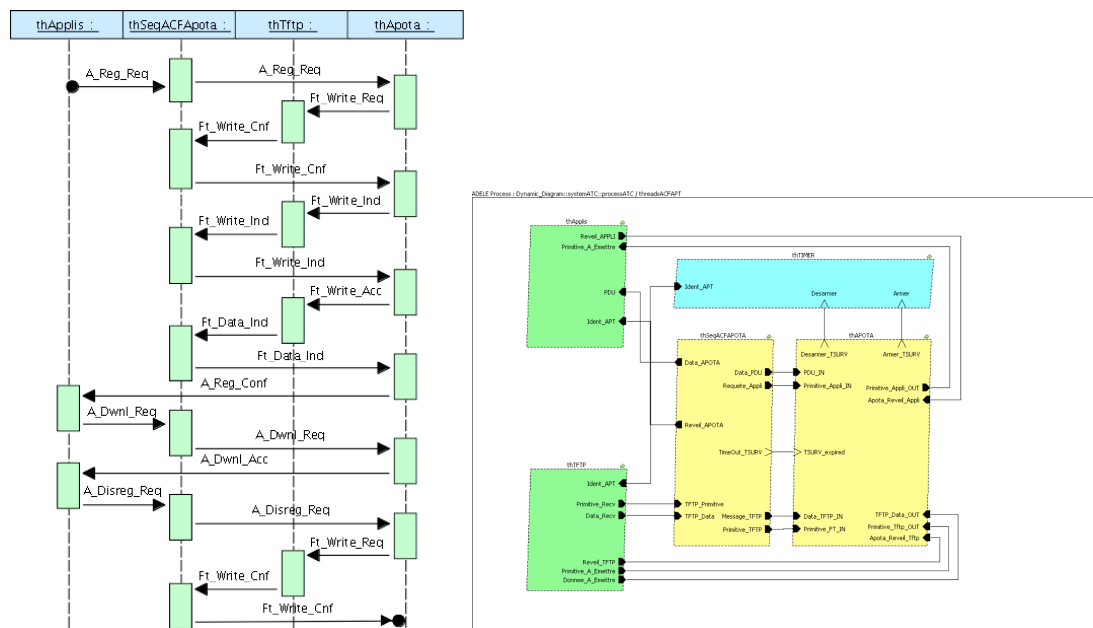


Simulink/Stateflow models

2.2.1.2 Airbus A380 IMA (Integrated Modular Avionics) software download protocol

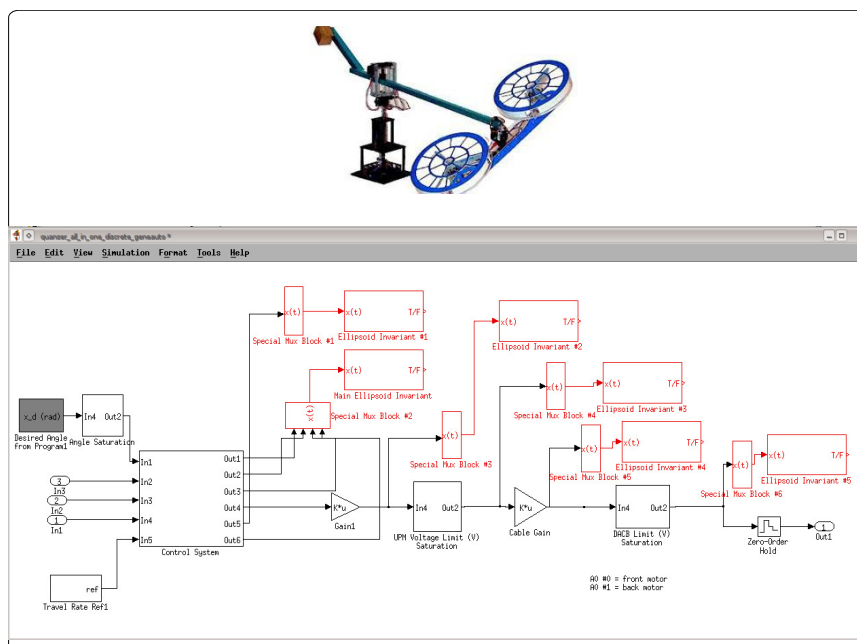
The models of an Airbus A380 IMA (Integrated Modular Avionics) software download protocol loosely derived from the TFTP (Trivial File Transfer Protocol) were developed in the TOPCASED, SPICES and Quartet projects using SAM/AADL/AADL-BA (mainly components and finite automata). The communication channel is error prone. The purpose of the validation is to assess that all states in the state machine are reachable.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	



2.2.1.3 Horizontal stability controller for a 3DOF (degree of freedom) Quanzer helicopter

The models of the horizontal stability controller for a 3DOF (degree of freedom) Quanzer helicopter developed in the CRAVES project using the Simulink/Lustre DSMLs (mainly dataflow). The black part of the following model represents the control algorithm. The red part represents observers that assess the quality of the algorithm. The validation purpose is to check that the algorithm is stable.



2.2.2 Languages used for these experiments

Finite state machine (or Finite automaton) and dataflow (and especially their Kahn process network interpretation)

Consortium restricted

© Consortium MOVIDA, 2012-2016

Page 11

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

are two models of computations that gave rise to several commonly DSMLs used by system engineers for control and command systems like fly by wire, auto pilot, etc. These DSMLs usually share the same abstract and concrete syntaxes but can behave in very different manners.

2.2.2.1 TOPCASED SAM: Structured Analysis Model

The SAM DSML was designed inside the TOPCASED project in order to express Structured Analysis Models loosely inspired from the HOOD and SA/RT methodologies. It focuses on the functional and logical architecture of the system control flow. It provides two kinds of diagrams: state machine and structured hierarchical interfaces.

2.2.2.2 SAE AADL: Architecture Analysis and Design Language

The Architecture Analysis & Design Language (AADL) is an architecture description language standardized by SAE (Society of Automotive Engineers). AADL was first developed in the field of avionics, and was known formerly as the Avionics Architecture Description Language.

The Architecture Analysis & Design Language is derived from MetaH, an architecture description language made by the Advanced Technology Center of Honeywell. AADL is used to model the software and hardware architecture of an embedded, real-time system. Due to its emphasis on the embedded domain, AADL contains constructs for modeling both software and hardware components (with the hardware components named "execution platform" components within the standard). This architecture model can then be used either as a design documentation, for analyses (such as schedulability and flow control) or for code generation (of the software portion), like UML.

AADL is defined by a core language that defines a single notation for both system and software aspects. Having a single model ease the analysis tools by having only one single representation of the system. The language specifies system-specific characteristics using properties.

The language can be extended with the following methods:

- User-defined properties: user can extend the set of applicable properties and add their own to specify their own requirements
- Language annexes: the core language is enhanced by annex languages that enrich the architecture description. For now, the following annexes have been defined.
 - Behavior annex: add components behavior with state machines
 - Error-model annex: specifies fault and propagation concerns
 - ARINC653 annex: defines modeling patterns for modeling avionics system
 - Data-Model annex: describes the modeling of specific data constraint with AADL

Inside this use case, we will consider a subset of instantiated AADL limited to the behavioral annex. This subset was defined by Airbus, Ellidiss and IRT in the TOPCASED, SPICES and quarteFt projects.

2.2.2.3 The MathWorks Simulink

Simulink, developed by MathWorks, is a data flow graphical programming language tool for modeling, simulating and analyzing multidomain dynamic systems. Its primary interface is a graphical block diagramming tool and a customizable set of block libraries. It offers tight integration with the rest of the MATLAB environment and can either drive MATLAB or be scripted from it. Simulink is widely used in control theory and digital signal processing for multi-domain simulation and Model-Based Design.

Inside this use case, we will consider a subset of Simulink defined inside the GeneAuto, EuroStars and P projects.

2.2.2.4 The MathWorks Stateflow

Stateflow, developed by MathWorks, is a control logic tool used to model reactive systems via state machines and flow charts within a Simulink model. Stateflow uses a variant of the finite-state machine notation established by David Harel, enabling the representation of hierarchy, parallelism and history within a state chart.[2] Stateflow also provides state transition tables and truth tables.

Stateflow is generally used to specify the discrete controller in the model of a hybrid system where the continuous dynamics (i.e., the behavior of the plant and environment) are specified using Simulink.

Specific applications for Stateflow include:

- Mode logic, where each discrete mode of a system is represented by a state
- Fault management, where the Stateflow chart is used to control how the system responds to faults and

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

failures within a system

- Task scheduling, where the Stateflow chart is used to schedule when specific tasks occur, either within the Stateflow chart or within the overall Simulink model

Inside this use case, we will consider a subset of Stateflow defined inside the GeneAuto, EuroStars and P projects.

2.2.2.5 Lustre/Esterel Technologies SCADE

Lustre is a formally defined, declarative, and synchronous dataflow programming language for programming reactive systems. It began as a research project in the early 1980s. In 1993 it progressed to practical, industrial use in a commercial product as the core language of the industrial environment SCADE, developed by Esterel Technologies. It is now used for critical control software in aircraft, helicopters, and nuclear power plants.

Inside this use case, we will not consider the graphical SCADE language but only the textual Lustre language used to give the semantics for SCADE.

2.2.2.6 OMG UML

Unified Modeling Language (UML) is a standardized (ISO/IEC 19501:2005), general-purpose modeling language in the field of software engineering. The Unified Modeling Language includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

The Unified Modeling Language was developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software in the 1990s.[1] It was adopted by the Object Management Group (OMG) in 1997, and has been managed by this organisation ever since. In 2000 the Unified Modeling Language was accepted by the International Organization for Standardization (ISO) as industry standard for modeling software-intensive systems. The current version of the UML is 2.4.1 published by the OMG in August of 2011.

Unified Modeling Language (UML) combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies.

UML 2.2 has 14 types of diagrams divided into two categories.[15] Seven diagram types represent structural information, and the other seven represent general types of behavior, including four that represent different aspects of interactions.

- Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems:
 - Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.
 - Component diagram: describes how a software system is split up into components and shows the dependencies among these components.
 - Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.
 - Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
 - Object diagram: shows a complete or partial view of the structure of an example modeled system at a specific time.
 - Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
 - Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.
- Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.
 - Activity diagram: describes the business and operational step-by-step workflows of components in

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

- a system. An activity diagram shows the overall flow of control.
 - UML state machine diagram: describes the states and state transitions of the system.
 - Use Case Diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.
- Interaction diagrams, a subset of behavior diagrams, emphasize the flow of control and data among the things in the system being modeled:
 - Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
 - Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.
 - Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
 - Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.

Inside this use case, we will mainly consider the subset of UML handled inside the P project (class, composite structure, activity and state machine diagrams).

2.2.3 Experimentation

The main purpose will be to assess the use of the GEMOC toolset in order to define xDSMLs.

Goal - The tools designed and implemented with the GEMOC studio should allow conducting Validation and Verification activities based on model execution (simulation) in a heterogeneous modeling environment. The GEMOC studio should ease the development of these tools.

Perimeter – The selected DSML families target late system engineering and early to late software engineering. The selection of several languages in the same family will allow experimenting the definition of language components at the various level of the DSML specification (AS, DSA, DSE, MoCC).

Expected Achievements - Methods and tools will be validated against existing simulation or verification tools for the selected subset of DSMLs:

- The GeneAuto2Polychrony simulator for Simulink/Stateflow models;
- The AADL2Polychrony simulator for AADL models;
- The AADL2Fiacre verifier for AADL models;
- The UML2PetriNet verifier for class, composite structure, state machine and activity diagrams;
- The TOPCASED UML model animator for class, composite structure, state machine and activity diagrams.

The GEMOC studio should ease the definition of several languages from the same family.

2.3 THALES: Functional Radar simulation

2.3.1 Application Overview

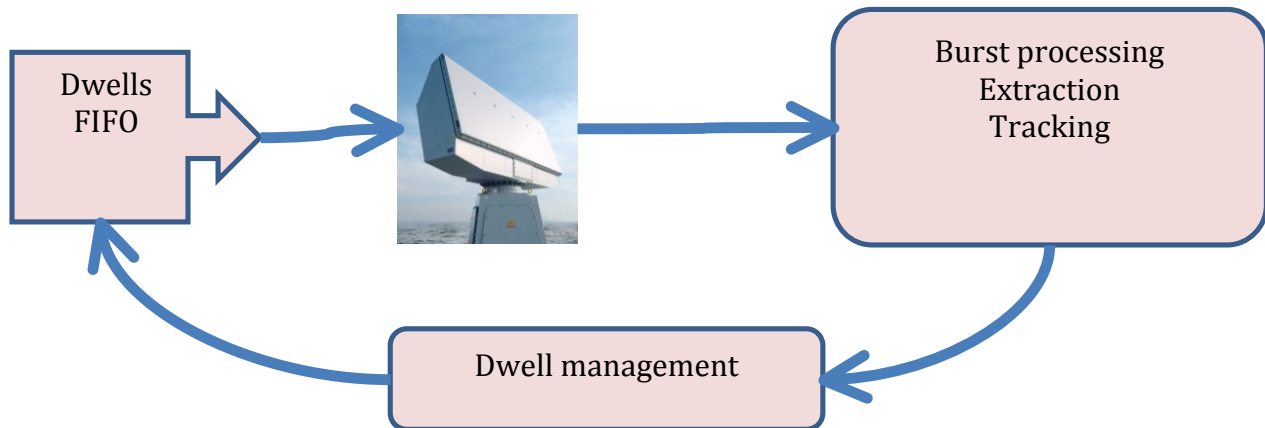
The goal of the application is basically to detect and follow flying objects in a nominal sector of sky. At a given moment, the radar observes a narrow angular sector in a direction by emitting a particular sequence of signal called dwell, and analyses the echo from this signal in order to detect the presence of objects (targets) and compute their distance and their speed. The radar must perform simultaneously two missions:

- A search mission where the whole sector is scanned exhaustively with the goal to detect new targets
- An active tracking mission where already detected targets are observed more frequently in order to enable tracking (showing their past trajectory and speed).

From the list of current tracks and the search mission schedule, the radar computes dynamically the direction and waveform of the dwells that must be sent one after the other by the antenna. On this point of view, the radar reacts

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

to a physical situation (flying objects) through a feedback control. This traditionally puts a particular importance on feedback delays which are here processing times and latencies.



It mixes several subsystems having different characteristics in terms of computation, communication, time or data. Hence, the heterogeneity that characterizes such a system makes the design and analysis really difficult. Besides, the development of such a system requires a good design space exploration through several levels of abstraction enabling good decision-making. Besides, such development requires managing the consistent refinement of the system model leading to a specification semantically rich enough to avoid snowballing errors and to allow relevant implementations.

The purpose of the radar application is to introduce some degree of dynamism, with state machine based control interacting with parallelisable, loop-based, sequences of processing.

The radar considered here has a fixed antenna and is able to send dwells in any direction within an overall 3D angular sector. Targets are simulated flying objects at various distances, direction, and speed and with various Radar Cross Sections (RCS) which drive their reflectivity. Within the wide observation sector is a protected area, needing more accurate observations of targets when they approach it. The effect of unfavorable weather conditions is also part of the simulation, with a rainy sector that adds unwanted spurious signals to the returned echo of the signals which cross it.

The first objective is to simulate the behavior in time of the radar in front of a scenario with many targets, taking account of processing latencies. The architecture and choice of appropriate MOC will be identified in the D5.1.2 deliverable.

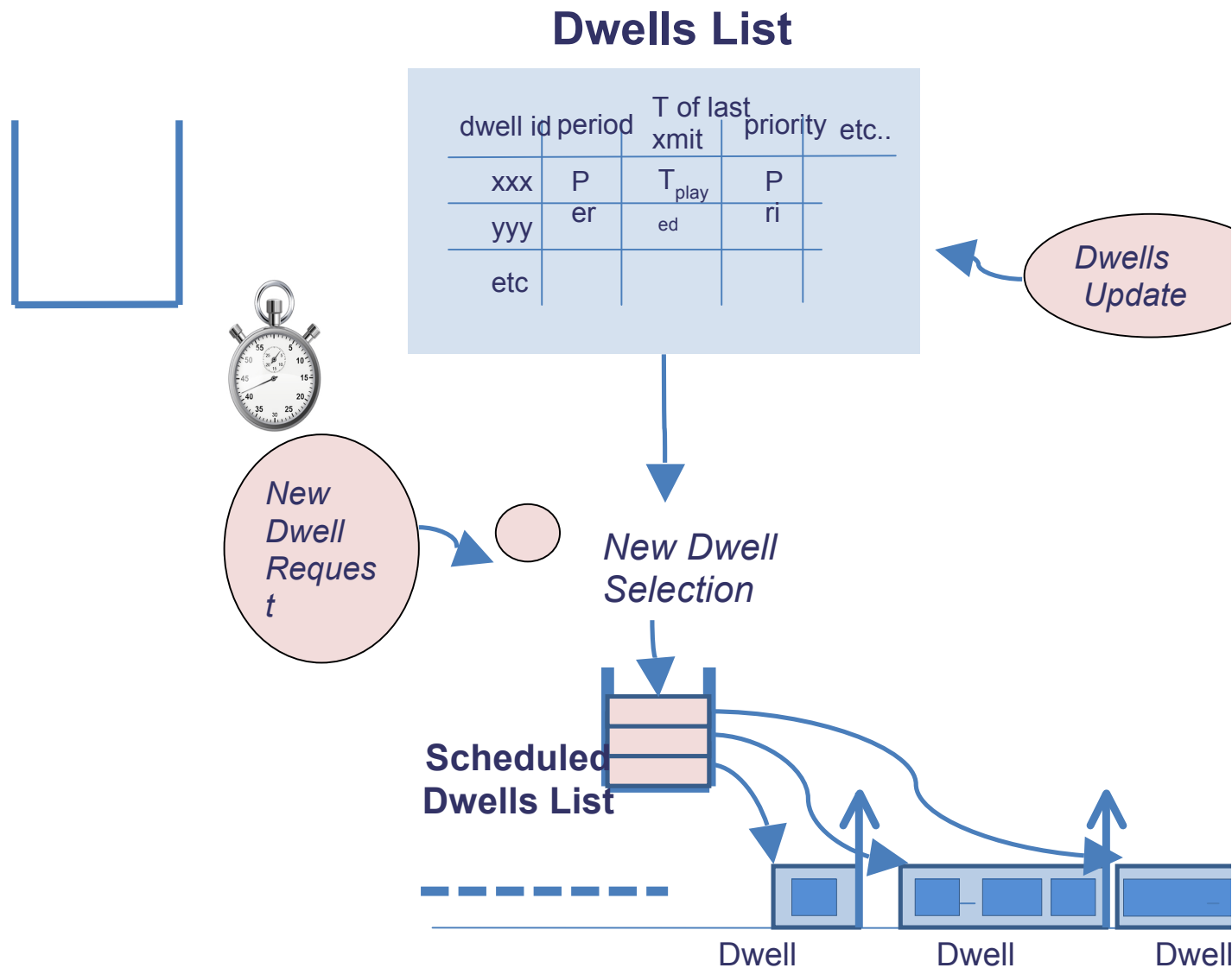
The following is an abstract of an existing detailed specification, where the computational contents of actors is skipped.

Extraction can fire after having received the hits of all the bursts of a dwell. A dwell contains one burst if it is an active tracking dwell, or 3 if it is a search dwell, or three bursts. Extraction has no memory of previous dwells. It makes some checks on the hits and produces zero, one or several plots to Tracking. A plot is attached to a particular direction in space (the direction of the dwell), so that plots are generally independent from each other. From a plot, Tracking updates a persistent table of tracks by creating, deleting or refining tracks. A track is, simplistically, the position and vector speed of a target. Following a track implies to illuminate _ i.e. send a dwell in the good direction. The time between two dwells sent for a track is critical in the sense that the direction of the next dwell is extrapolated from the estimated speed of the track.

Dwell management closes the loop by controlling the sequence of dwells/bursts sent sequentially through the antenna. It manages a list of candidate dwells with all the search dwells (those not attached to tracks) and the active tracking dwells. The selection of a candidate among others is done first by its level of priority, and by an indicator of how long ago it was selected with respect to its nominal activation period. The list of dwells needs regular updates. A candidate dwell is pushed into the scheduled dwells fifo. A dwell has a variable number of bursts (1 or 3) and each burst has a variable duration (type any between 10 to 20 ms). The fifo is

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

read by the antenna each time a dwell has finished to be transmitted, and the new dwell is sent to Echo which issues some inputs to Burst.



The processing latency delays between inputs and outputs of actors will in general be a function of some dynamic parameters evaluated at runtime, such as number of plots, burst duration, etc...

2.3.2 ARCADIA (ARCHitecture Analysis & Design Integrated Approach)

In order to build architecture of a software intensive system, many stakeholders contribute to the description of the system architecture. Following a model-based engineering approach, the different stakeholders will use modeling tools to describe the architecture and analysis tools to evaluate some properties of the architecture.

Thales has defined a system and software architecture engineering approach, named ARCADIA (ARCHitecture Analysis & Design Integrated Approach)[ref]. This approach is based on architecture-centric with model-driven

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

engineering activities and aims at finding the best architectural solution given the systems constraints. The ARCADIA approach uses a viewpoint-based architectural description, such as described in the conceptual foundations of ISO/IEC 42010 Systems and Software Engineering - Architecture Description [ref]. Viewpoints are the way that each stakeholder sees the same part of the system. In Thales scenario, these viewpoints are related to the system architecture. Therefore, we can identify two main dimensions in Thales model-based system development cycle, which are illustrated in the Figure ARCADIA overview. The first dimension is the set of engineering and architecture definition phases, and the second is the set of views. In this work, as a concrete example, we present the five steps/phases that follow the model-based paradigm and that can benefit from VM modeling, which are the ones related to the systems architecture definition. Following are each model-driven phases, with their respective titles used in Thales, and a brief description.

Customer Operational Need Analysis: This step focuses on analyzing the customer needs, goals, expected missions and activities. This step ensures that the system definition is adequate to its real operational use. This phase also defines the IVVQ (Integration, Verification, Validation and Qualification) conditions.

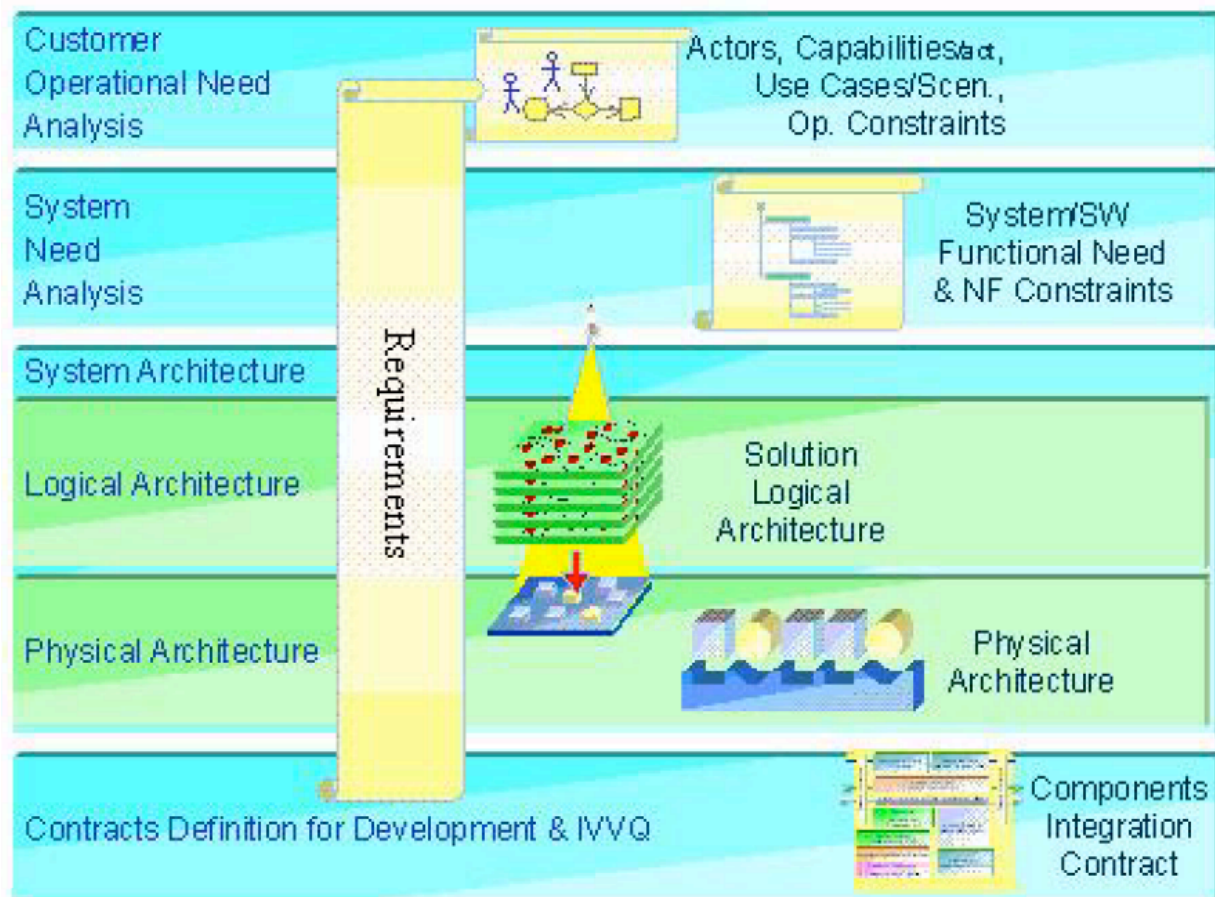
System Need Analysis: This step focuses on defining how the system can satisfy the operational need, along with its expected behavior and qualities.

System Logical Architecture Definition: This step focuses on identifying the system parts, their contents, relationships and properties, excluding implementation or technical/technological concerns. In order to assure that these parts are stable and safe to the further steps, all major non-functional constraints, such as: safety, security and performance, are taken into account to find the best trade-off among them.

System Physical Architecture Definition: This step also focuses on defining the system architecture, but in the physical components level, making them ready to be developed in a low-level engineering.

Definition of Components Development & IVVQ technical Contract EPBS: This step focuses on supporting the construction of the EPBS (End-Product Breakdown Structure), benefiting from the previous architectural definition, defining components requirements, and preparing a safe IVVQ (Integration, Verification, Validation and Qualification).

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	



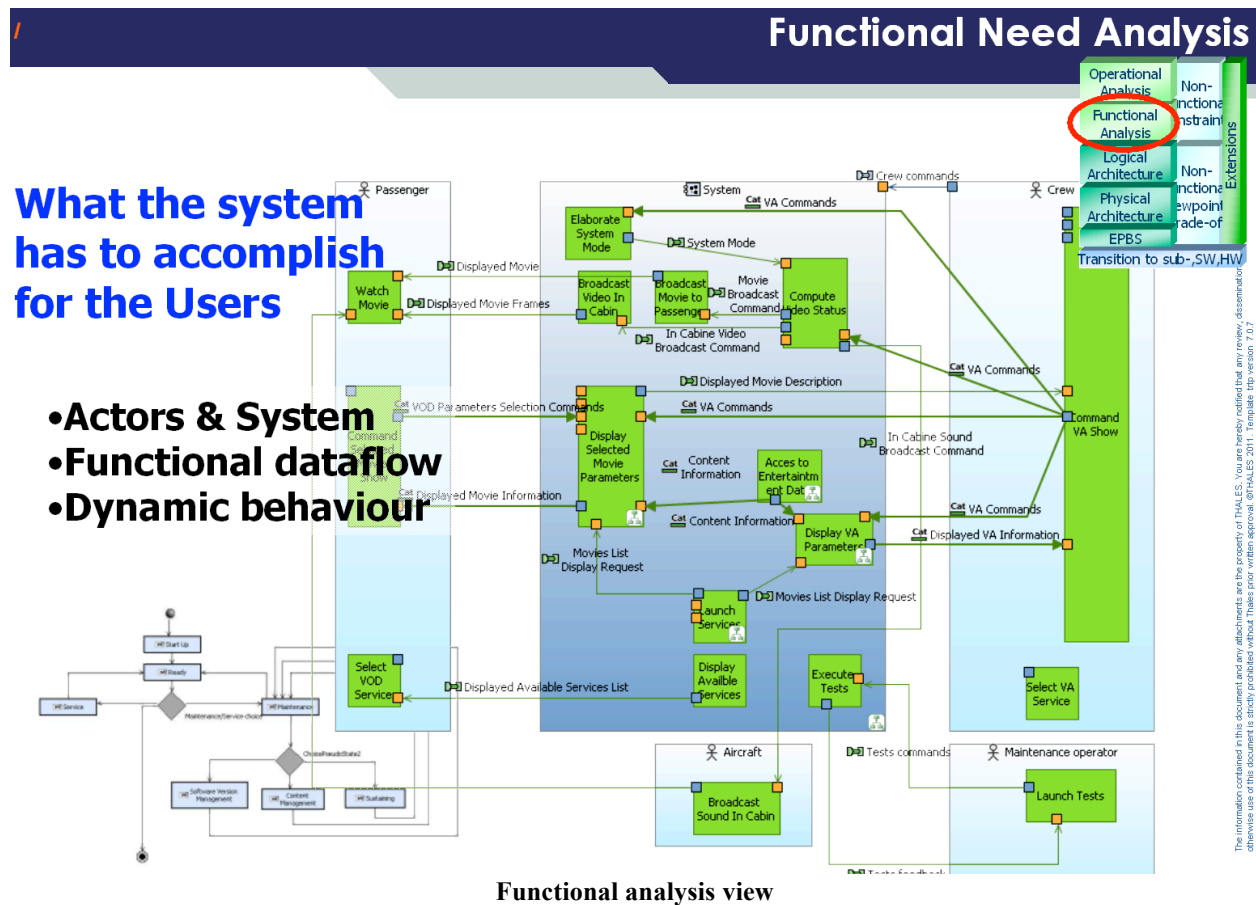
ARCADIA Overview

2.3.3 Abstract Syntax & Concrete syntax

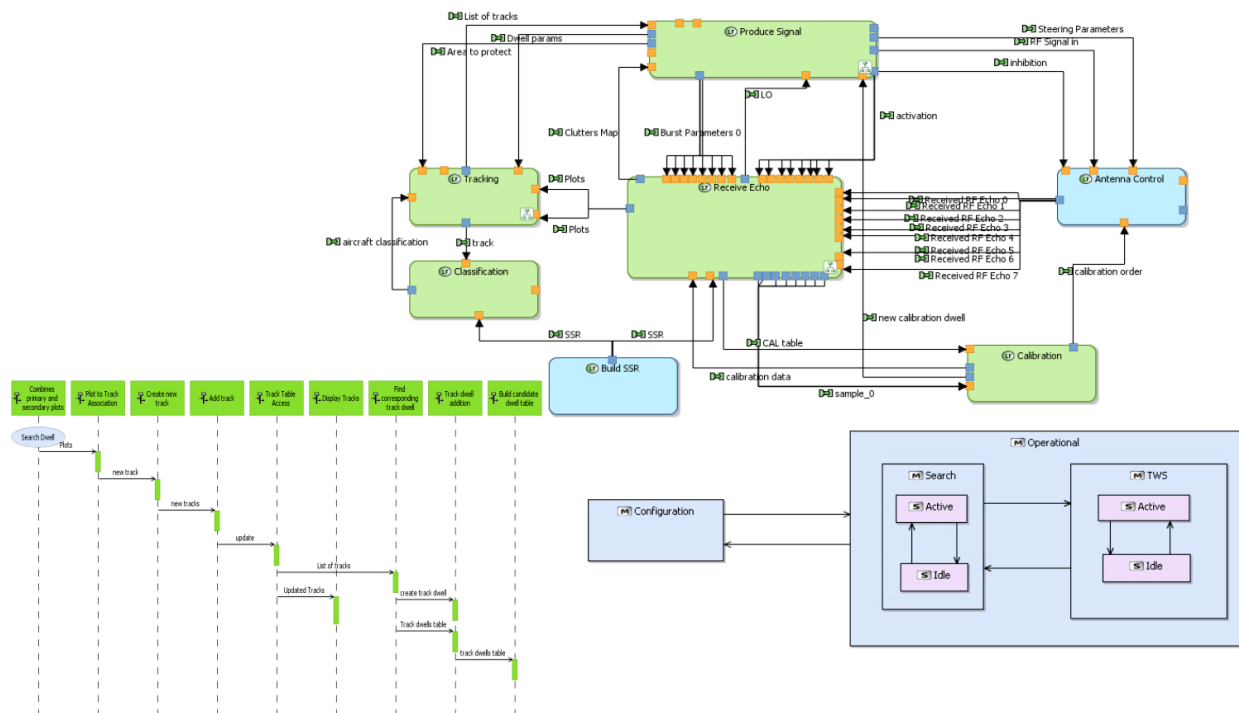
The engineering meta-model defined by Thales is composed of a set of 20 meta-models and about 400 meta-classes involved in the five viewpoints (a.k.a Architecture level) defined in ARCADIA.

Our experiment environment consisted of a System engineering tool and the GEMOC studio. The graphical notation of our engineering workbench has been defined thanks to the Eclipse Sirius project. It allows engineers to define the architecture description of a software system by providing the five “views”. In the context of GEMOC we will focused on the two following views, System analysis and logical architecture. The two figures below show the different concrete syntaxes used for the two mentioned architecture level.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	



WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	



Logical Architecture view

2.3.4 Behavioral Semantics

The available way to model behavior is limited to the description of modes and states, specification of functional chains and scenarios without coordination of these different languages.

Overcoming current limitations

To support the execution of models, we have to define a dedicated *Execution Model* for the three both part of the Thales language.

In addition, to capture the behavioral semantics, the interpretation needs to define how the execution of the scenario, data flow and state machine proceeds over time.

2.3.5 Experimentation

Goal - The study will propose methods and tools to support System Engineers to tackle system modeling activity in order to improve, at upper analysis abstraction levels, confidence in the specification of the system to be built. Purpose is both to reduce risks concerning functional requirements and to give drivers to design the system architecture. It is realized by “executing” through a simulation the existing specification and by checking it through the analysis of the simulation.

Perimeter - All System Engineers dealing with complex systems definition are targeted by the proposed methods and tools which were designed to be compliant with Melody-Advance and moreover to fully capitalize on its already

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

existing features. *Modeling extensions to describe behavior at a high level require a limited effort to be added and are fully compliant with lower system analysis / definition abstraction levels.* They allow one to validate, at upper analysis abstraction levels, a system definition according to a set of relevant early-validation criteria taking into account the balance between modeling effort and pertinence of the results.

Expected Achievements - Methods and tools will be validated through an integrated advanced simulation prototype in the Thales system engineering against the radar system.

The experimentation will be focused on the following topics at M1 level:

1. achievable validation goals using a simulation during the system analysis abstraction level,
2. behavior modeling needs to get those validation goals,
3. hypothesis choices that allow to perform simulation,
4. exploitation of simulation results,
5. validation of the approach.

The experimentation will be focused on the following topics at M2 level:

1. Use the global MoCCs and xDSMLs design methodology for defining :
 - a. the mode & states and data flow execution semantics,
 - b. implementation of the relevant MoCCs for the radar use case,
 - c. the mode & states and data flow coordination.
2. Customize the graphical notation for animation of each language.
3. Integration the execution framework in the Thales engineering workbench.

Any validation phase means first to set a list of main drivers to be checked to get confidence in the work already performed and to get a go ahead for the next steps of the process.

We have defined some relevant objectives with associated validation criteria are listed below for the validation of the specification of a system at upper analysis abstraction levels.

Identified early-validation objectives can be categorized in two sets depending if they are considered or not as relevant at system analysis abstraction level.

All objectives are relevant at lower analysis abstraction levels, a study on that abstraction levels would benefit from the works realized.

Relevance of objectives for each abstraction level	Functional requirements completeness and consistency validation	Activation order checking	System states checking	Performance requirements validation	Coverage checking	Exchanges activation frequency validation	Buffer size validation of pending exchanges	Functional chain validation	Giving help for sizing	Data availability validation	Coherency of data to synchronize checking
System analysis	✓	✓	✓	✓	✓	✓	✓	✓	✗	✗	✗
Logical architecture	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Physical architecture	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓




WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

To be able to cover each defined objective, different types of simulation, with different levels of modeling needs, have to define.

For the objectives that are covered both by functional chain simulation and behavior simulation types, validation is more valuable using the second simulation type.

Relevance of objectives for simulation types at system abstraction level	Functional requirements completeness and consistency validation	Activation order checking	System states checking	Performance requirements validation	Coverage checking	Exchanges activation frequency validation	Buffer size validation of pending exchanges	Functional chain validation
Functional chains simulation	✗	✓	✓	✓	✓	✓	✓	✗
Behavior simulation	✓	✗	✓	✓	✓	✓	✓	✗

3. Technical requirements

Requirement Identifier	Description / comment	Result   
[GEMOC-ALL-REQ-001]	The GEMOC studio shall provide a xDSML Process	
[GEMOC -ALL-REQ-002]	The GEMOC studio shall provide means to define <i>Executable DSML</i>	
[GEMOC -ALL-REQ-003]	The GEMOC studio shall provide means to define Model of Computation (MoC), which is a set of rules defining the semantics of control and concurrency, the semantics of communications and the semantics of time of the modeling paradigm.	
[GEMOC-ALL REQ-004]	The GEMOC studio shall provide means to define textual concrete syntax for EMF metamodel based DSML	
[GEMOC-ALL REQ-005]	The GEMOC studio shall provide means to simulate a model	
[GEMOC-ALL REQ-006]	The GEMOC studio shall provide means to animate a model	
[GEMOC-ALL REQ-007]	The GEMOC studio shall provide means to define diagram layers and for filters for the animation	
[GEMOC-ALL REQ-008]	The GEMOC studio shall provide means to define table	

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

4. Metrics for Evaluation

At this stage of the project, we have identified the relevant concerns that will guide the GEMOC studio's evaluation. The evaluation will be conducted according to the GEMOC Language definition process. Experimentation of use case providers will provide metric later according to the following concerns:

Concern #1: Static Semantics Definition

The static semantics of a DSL is the union of the well-formed rules on top of the abstract syntax (as invariants of domain classes) and the axiomatic semantics (as pre- and post-conditions on operations of metamodel classes). The static semantics is used to statically filter incorrect DSL programs before actually running them. It is also used to check parts of the correctness of a DSL program's execution either at design- time using model-checking or theorem proving, or at run- time using assertions, depending on the execution domain of the DSL. Kermeta uses the OMG Object Constraint Language (OCL) to express the static semantics, directly woven into the metamodel using the Kermeta aspect keyword.

In the Kermeta workbench, the abstract syntax and the static semantics are conceptually and physically (at the file level) defined in two different modules. Consequently, it is possible to define different semantic variants for the same domain, i.e. to have a single EMOF metamodel shared by different static semantics, e.g., to cope with language variants.

Concern #2: Behavioral Semantics Definition

The GEMOC studio shall provide means to define of the behavioral semantics of a DSL. To define the behavioral semantics of a DSL, the GEMOC studio provide the Kermeta Language, an action language that is used to express the behavioral semantics of a DSL. It can be used to define either a translational semantics or an operational semantics. A translational semantics would result in a compiler while an operational semantics would result in an interpreter. In this study, we will evaluate only the operational semantics capability definition.

Using the Kermeta language, an operational semantics is expressed as methods of the classes of the abstract syntax. The body of the method imperatively describes what is the effect of executing an activity.

Concern #3: Model of concurrency Definition

The GEMOC studio shall provide means to define Model of concurrency.

Concern #4: Graphical Syntax Definition

The GEMOC studio shall provide means to define of the Graphic syntax thanks to the Eclipse Sirius project. This project provide :

- The ability to define workbenches providing editors including diagrams, tables or trees.
- The ability to integrate and deploy the aforementioned environment into Eclipse IDE's or RCP applications.
- The ability to customize existing environments by specialization and extension.

From the end user perspective, Sirius provides:

- Rich and specialized modeling editors to design their models.

WP4 – Workbench Set-up Experimentation / Task 5.1	Version: 1.0
D 4.1 Test Case Specification	Date: 04/06/2013
62441757/306/1/-	

- Synchronization between these different editors.

Concern #5: Animation

The GEMOC studio shall provide means to extend the Graphic syntax for animation purpose.