Grant ANR-12-INSE-0011

GEMOC

ANR Project, Program INS

WP5 – GEMOC EXPERIMENTATIONS

D5.1.1 Technical Requirements, uses-cases specification and metrics for experimentations Task 5.1

Version 1.1

DOCUMENT CONTROL

	1.0 :	1.1 :	:	:	:
Written by Signature	Jérôme Le Noir (TRT)	Jérôme Le Noir (TRT)			
Approved by Signature					

Revision index	Modifications
A	TRT and IRIT use case studies have been updated
B	
C	
D	

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

Table of content

D5.1.1 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 meta-modelling pattern to build xDSML in order to define:

- Modelling languages with associated methods and tools for the modelling 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 excepted 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

- **AS:** Abstract Syntax.
- **API:** Application Programming Interface.
- **Behavioral Semantics:** see **Execution semantics.**
- **CCSL: Clock-Constraint Specification Language**.
- **Domain Engineer:** user of the Modeling Workbench.
- **DSA:** Domain-Specific Action.
- **DSE:** Domain-Specific Event.
- **DSML:** Domain-Specific (Modeling) Language.
- **Dynamic Semantics:** see **Execution semantics.**
- **Eclipse Plugin:** an Eclipse plugin is a Java project with associated metadata that can be bundled and deployed as a contribution to an Eclipse-based IDE.
- **ED:** Execution Data.
- **Execution Semantics:** Defines when and how elements of a language will produce a model behavior.
- **GEMOC Studio:** Eclipse-based studio integrating both a language workbench and the corresponding modeling workbenches.
- **GUI:** Graphical User Interface.

- **Language Workbench:** a language workbench offers the facilities for designing and implementing modeling languages.
- **Language Designer:** a language designer is the user of the language workbench.
- **MoCC:** Model of Concurrency and Communication.
- **Model:** model which contributes to the convent of a View.
- **Modeling Workbench:** a modeling workbench offers all the required facilities for editing and animating domain specific models according to a given modeling language.
- **MSA:** Model-Specific Action.
- **MSE:** Model-Specific Event.
- **RTD:** RunTime Data.
- **Static semantics:** Constraints on a model that cannot be expressed in the metamodel. For example, static semantics can be expressed as OCL invariants.
- **TESL:** Tagged Events Specification Language.
- **xDSML:** Executable Domain-Specific Modeling Language.

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.

1.4 References

- GEMOC Document Scientifique
- GEMOC D4.1.1 “GEMOC architectural description”
- GEMOC D4.1.2 “GEMOC studio, v0.1”

1.5 Summary

This document defines the context in which the GEMOC studio prototyping results will be evaluated. It provides the description of the excepted 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.
Finally , Section 3 identifies requirements which will be used for the evaluation of the GEMOC studio.

2. Uses-cases specification

Several aspects must be considered in GEMOC related to the coordination of heterogeneous Domain Specific Modelling 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 behavioural semantics of systems. It is computationally complete by specifying a full behavioural 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.

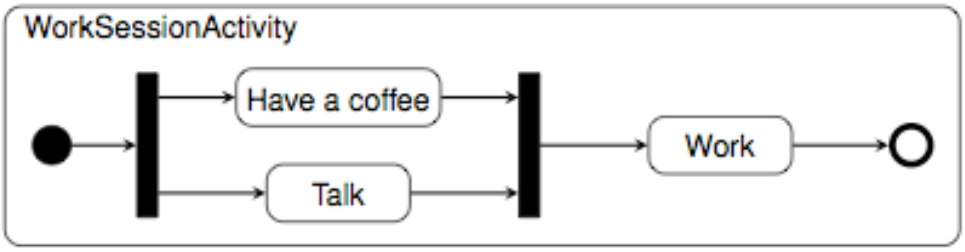


Figure 1 : Activity of the members of our team during our work sessions

The fUML specification includes both a subset of the abstract syntax of UML, and an execution model of that subset supported by a behavioural 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.

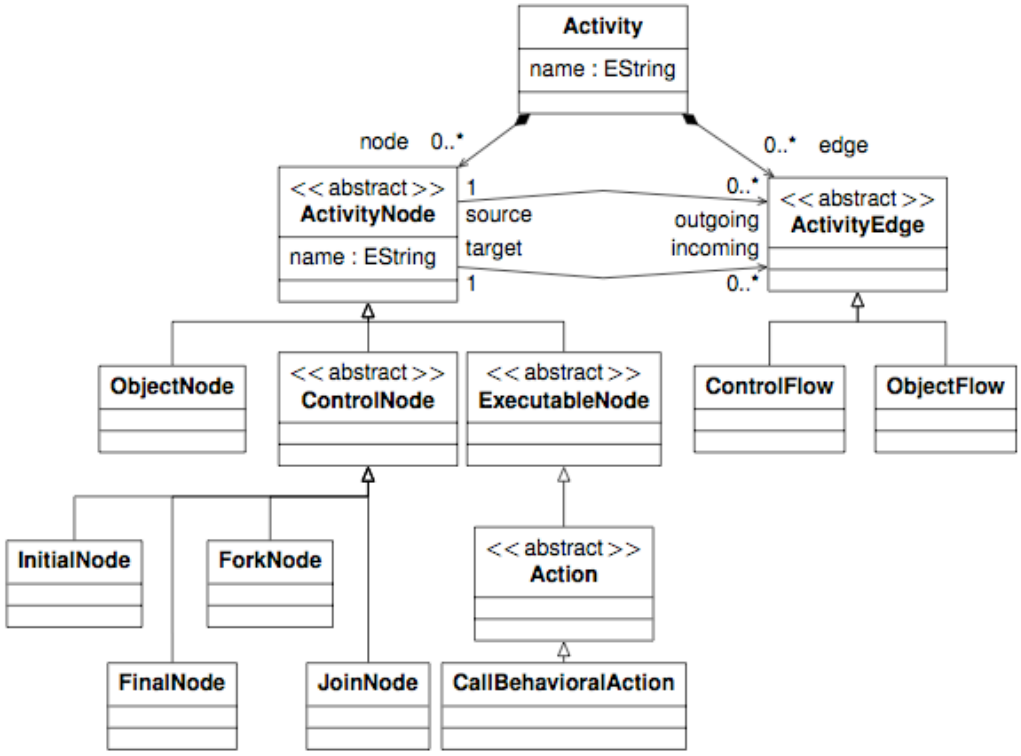


Figure 2 : Excerpt of the fUML Metamodel

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 modelled 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

visitors.

In addition, to capture the behavioural 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 behaviour 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 behavioural 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 behavioural 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 Concurrency and communication* (MoCC) of the language. Such an implementation prevents the reuse of a MoCC for different DSMLs (e.g., the fact that all the domain-specific actions should run in sequence is a behavioural 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 MoCC. 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".

2.2 IRIT: Coordination of different semantic variants for the same DSML Use Case

These experiments target the specification of various executable Domain Specific Modelling languages (xDSMLs) from the same family with different semantics variants. The main purpose is to assess the capability to share as much as possible the common parts and only provide the minimal mandatory semantic specific elements. The chosen xDSMLs, from the data flow and state machine paradigms, 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. We give several motivating scenario mainly derived from Airbus needs that illustrate various potential use of the GEMOC toolset in order to broaden the scope of the project and provide other requirements than the ones from THALES Research and Technologies. We assist Airbus in the development of one of these scenario and we develop on our own another one.

2.2.1 Motivating scenarios

These scenarios rely on different simple yet realistic subsystems extracted from previous use cases developed in other projects. It also relies on collaboration between Airbus and IRIT to fuel the GEMOC projects with additional industrial requirements. The associated work inside GEMOC for these use case is mainly 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 Maintenance training scenario validation (in cooperation with Airbus)

This scenario targets the development by Airbus of training simulators for maintenance team. The purpose is to ease the learning of plane diagnosis and repair procedures. In order to develop such simulators, an initial phase is conducted to elicit instructor requirements as learning objectives and pedagogic scenarios. This phase must be agile and several iterations must be conducted in order to obtain the appropriate data. Simulation tools would help in being more agile, ease the communication between development teams and instructors, allow the early validation of scenarios, and assess the compatibility of scenarios with the learning platforms.
The main purpose is thus to validate with the instructors their learning scenarios for the training of plane maintenance team before their implementation on simulators.
The behaviours of the planes are expressed using the Simulink/Stateflow languages. The learning scenarios (pedagogic side) are given using a subset of SysML/UML activity diagrams. The behaviors of the maintenance teams (through the simulator GUIs) are given as state machines (a subset of SysML/UML state machine diagram).
This scenario has been selected as the most representative use case by Airbus for conducting an experiment using

the GEMOC studio.

2.2.1.2 Airbus AP2633 standard formal modelling (in cooperation with Airbus)

Airbus is relying on the OCASIME framework to develop integration simulation tools. The purpose is to assess the compatibility of the various parts of a full system. These parts are first simulated using the specification and design models. Then each model is replaced by the operational subsystem when it becomes available and the behaviour preservation is assessed in a co-simulation manner. In order to allow the collaborative simulation of the various models and operational parts, Airbus has designed the cooperative middleware AP2633 that allows to share data between parts and to coordinate the execution of the various parts. Currently, AP2633 is only available as a natural language specification and several implementations that are difficult to assess with respect to this industrial standard. The purpose of this scenario is to rely on the GEMOC studio to formally specify AP2633 as a MoCC and to express the cooperation between the various languages as composition operators.

2.2.1.3 Hardware/Software timing requirement validation (in cooperation with Airbus)

This scenario targets the development of safety critical real time systems involving specific hardware. The targeted part is the ARINC 429 point-to-point communication bus. It provides a dedicated hardware to encode/decode data packet that can be observed either by polling or interrupts. However, interrupts are forbidden to avoid complex timing behaviour, and polling has a too heavy computation cost. Thus, the software must issue request in a timely manner that takes into account the worst-case coding/decoding time. Simulation is used in order to verify the correct behaviour of the hardware/software integration, i.e. that the request to the hardware are always sent when this one has finished encoding/decoding the data from the previous requests. The development of models has been conducted by Airbus in the last five year in an experimental project targeting a full model based development of a fly by wire system. The models are expressed in a subset of SysML/UML state machine and activity diagrams for the parts that are currently hand coded in real systems whereas the other parts are modelled in SCADE and autocoded. This verification is conducted at each step during the detailed design and implementation phases to ensure non-regression.

2.2.1.4 Door Management System early model validation and refinement verification (in cooperation with Airbus)

Safety systems must ensure that a plane doors cannot be opened when the cabin pressure in too high with respect to outside of the plane. This feature is strongly regulated in the avionics domain. The purpose of this scenario is first to ensure that the requirements are correct with respect to the designer purpose, and that the early design of the architecture is a refinement of the requirements. The models for the Door Management System of the Airbus A350 were developed in the CESAR project using the AADL/Simulink/Stateflow xDSMLs (mainly component and state machine diagrams). The simulation assesses 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 using the Simulink/Stateflow languages. It is a requirement model from the functional architecture phase to the logical architecture phase. Then, the architecture of the system is designed using the AADL and the behaviour of each part is designed using the Simulink/Stateflow languages. The verification aims at assessing that both models behave in the same way. It was conducted in CESAR using the PolyChrony toolset and a translation from Simulink/Stateflow to PolyChrony based on GeneAuto, and from AADL to PolyChrony.

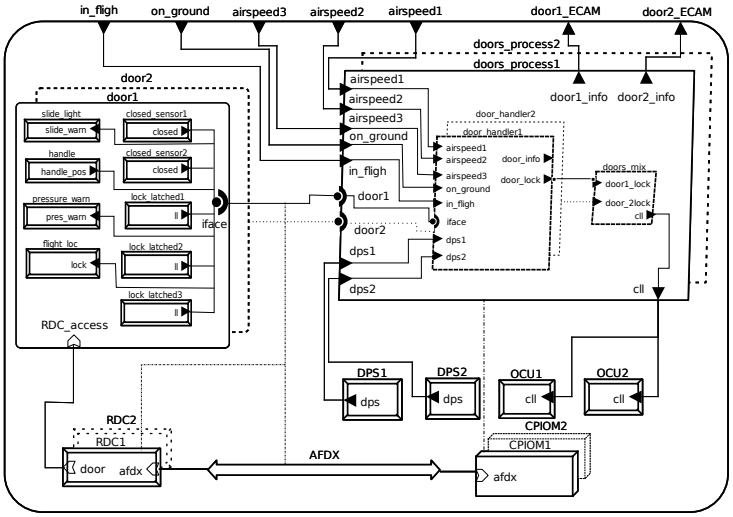


Figure 3 : AADL Models

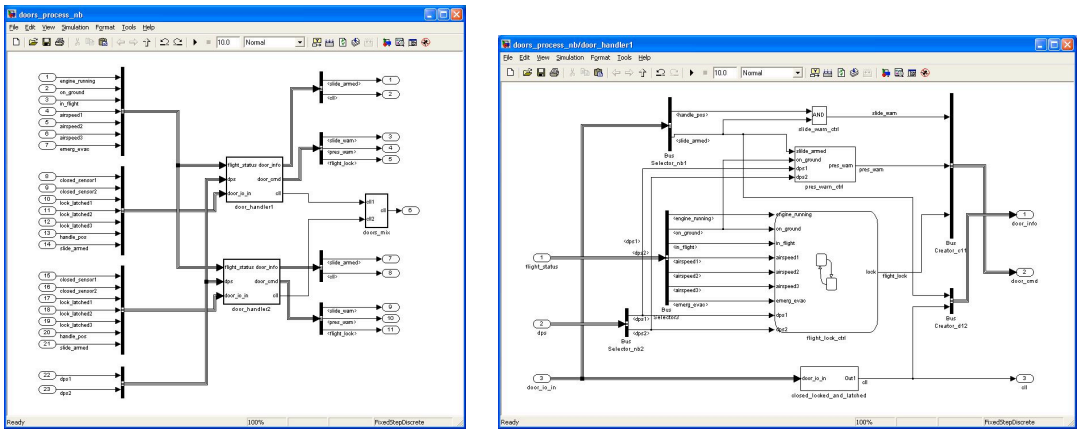


Figure 4 : Simulink/Stateflow model

2.2.1.5 A380 IMA (Integrated Modular Avionics) software download protocol (in cooperation with Airbus)

The purpose of this scenario is to ensure the correctness of protocols with respect to generic requirements like “each state of the protocol can be accessed in a finite time”. The application is a part of the Airbus A380 IMA (Integrated Modular Avionics) software download protocol loosely derived from the TFTP (Trivial File Transfer Protocol) standard. One key point is that the communication channel is error prone. The models were developed in the TOPCASED, SPICES and quarteFt projects using the AADL and its behavioural annex (mainly component and state machine diagrams). The purpose of the validation is to assess that all states in the state machine are reachable. The validation is conducted using several hypotheses regarding the behaviour of communication channels. It was conducted previously using a translation from AADL to the Fiacre language and the TINA model checker.

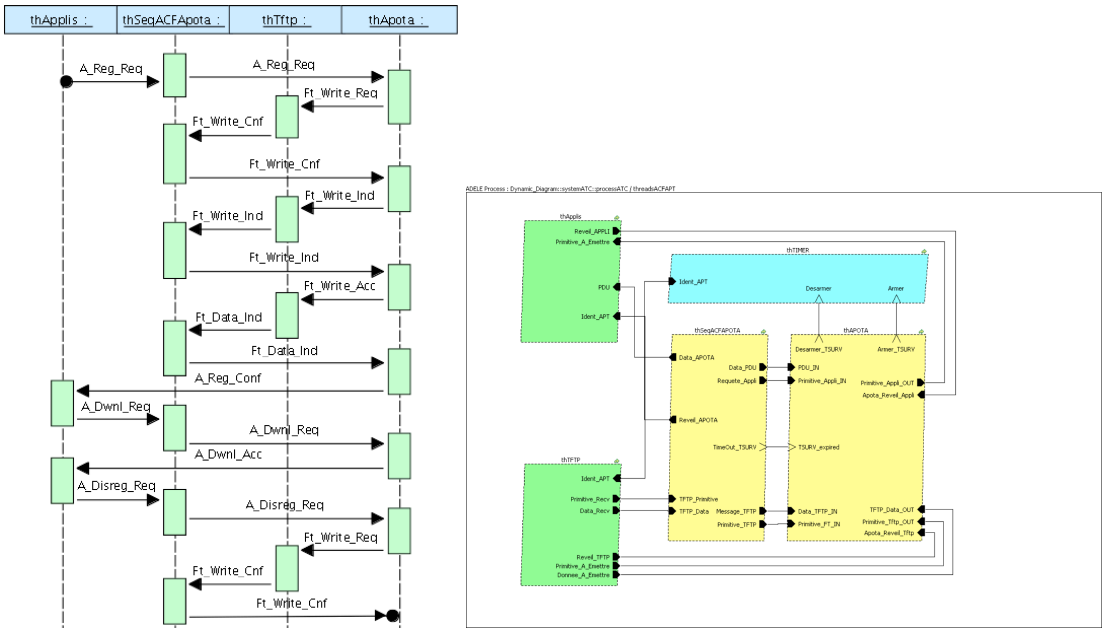


Figure 5 : AADL Models

2.2.2 Experiments conducted in the GEMOC project

Based on these scenarios, IRIT in cooperation with Airbus conducts the two following experiments in language design in order to fuel the requirements and design of the GEMOC studio.

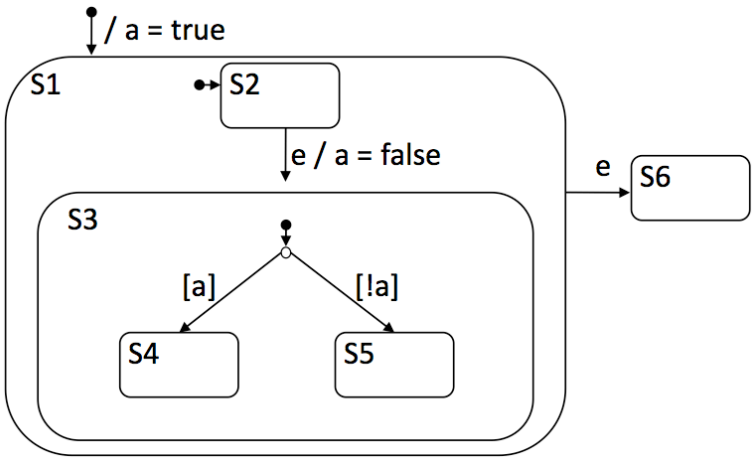


Figure 6 : State machine semantic variations (courtesy of G. Karsai et al.)

2.2.2.1 Language design use case: Management of state machine semantics variation

Several variants of the state machine semantics are used in the various scenarios: AADL mode automata, AADL behavioural annex, Stateflow, UML and several others are commonly used like SCADE, Statemate, Rhapsody, Esterel, etc. All these variants share the same concepts like State, Junction, Transition, Activity, Action, Guard, Regions, Charts, etc. Parts of their semantic are also similar. However, Karsai has shown that the execution of the same diagram given in Figure 6when handling event `e` with the semantics from UML, Rhapsody and Stateflow

leads to three different states `S4` (UML), `S5` (Rhapsody) and `S6` (Stateflow). The first purpose of this experiment is to implement several semantics and to obtain the same results as Karsai. This fist step makes sense, as state machine is a very common modelling language that is used in many different domains and development phases with many different purposes. There is thus a global understanding of the meaning of state machine and many different semantic variants. This has been illustrated by Harel when he designed StateCharts with a synchronous (StateMate tool) and asynchronous (Rhapsody tool) and by the OMG in the UML standard that provides many Semantics Variation Points for the state machine diagram. The second purpose is then to combine in the same diagram the various languages and be able to predict the behaviour of the model. This second step makes sense as all systems are currently developed using different languages for different aspects: mode automaton, protocol and behaviour state machine, both synchronous and asynchronous. The following semantics are currently considered: Stateflow, AADL mode automaton, AADL behavioural annex, Rhapsody state machine, UML state machine.

2.2.2.2 Airbus maintenance scenario validation

This use case has been defined in section 2.2.1.1. It relies on the GeneAuto/Projet P/HiMoCo subset of Simulink/Stateflow, the subset of the UML state machine diagram developed in the previous use case and the subset of fUML developed in the INRIA use case. The DSA for the subset of Simulink/Stateflow are generated from the formal specification of the GeneAuto block library developed by IRIT in the FUI Projet P/HiMoCo projects.

2.2.3 Languages used for these scenarios and use cases

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

2.2.3.1 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 eases 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: annex languages that enrich the architecture description enhance the core language. For now, the following annexes have been defined.
    - Behaviour annex: add components behaviour with state machines
    - Error-model annex: specifies fault and propagation concerns
    - ARINC653 annex: defines modelling patterns for modelling avionics system
    - Data-Model annex: describes the modelling of specific data constraint with AADL

In these scenarios, only a subset of instanciated AADL limited to the behavioural annex has been used. Airbus, Ellidiss and IRIT in the TOPCASED, SPICES and quarteFt projects defined this subset.

2.2.3.2 The MathWorks Simulink

Simulink, developed by The MathWorks, is a data flow graphical programming language tool for modelling, 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.
In these scenarios, only a subset of Simulink defined inside the GeneAuto, HiMoCo and P projects has been used.

2.2.3.3 The MathWorks Stateflow

Stateflow, developed by The 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 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

In these scenarios, only a subset of Stateflow defined inside the GeneAuto, HiMoCo and P projects has been used.

2.2.3.4 OMG UML

The Unified Modeling Language (UML) is a standardized (ISO/IEC 19501:2005), general-purpose modelling 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 modelling 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 modelling (entity relationship diagrams), business modelling (work flows), object modelling, and component modelling. 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 behaviour, including four that represent different aspects of interactions.

- Structure diagrams emphasize the things that must be present in the system being modelled. 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 artefacts deployed on the hardware.
    - Object diagram: shows a complete or partial view of the structure of an example modelled 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.

- Behaviour diagrams emphasize what must happen in the system being modelled. Since behaviour diagrams illustrate the behaviour 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 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 behaviour diagrams, emphasize the flow of control and data among the things in the system being modelled:
    - 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 behaviour 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.

In these scenarios, only the subset of UML handled inside the P project (class, composite structure, activity and state machine diagrams) has been used.

2.2.4 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 modelling 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 execution of the code generated by the GeneAuto/Project P ACG;
- 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.

Planning – The MathWorks Simulink will be first experimented for the M+18 deliverable as it is a key element for all use cases. It will first be hand-coded then the DSA will be automatically generated from the block library specification in the P/HiMoCo projects. The MathWorks Stateflow will only be experimented when the P/HiMoCo projects importer will be available at M+24 (this one has been delayed in P/HiMoCo projects). The UML activity diagram part will rely on the fUML use case and the UML state machine part and AADL part will be integrated with the Stateflow part at M+24.

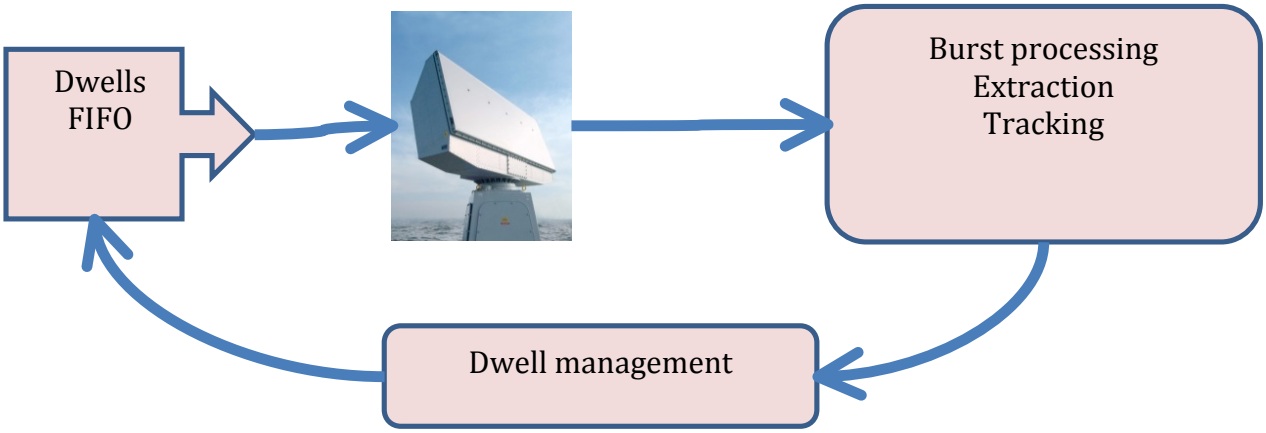
2.3 THALES: Radar simulation

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 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 behaviour in time of the radar in front of a scenario with many targets, taking account of processing latencies. The architecture and choice of appropriate MoCC will be identified in the D5.2.1 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 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 modelling 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 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. 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 modelling, 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 analysing 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.

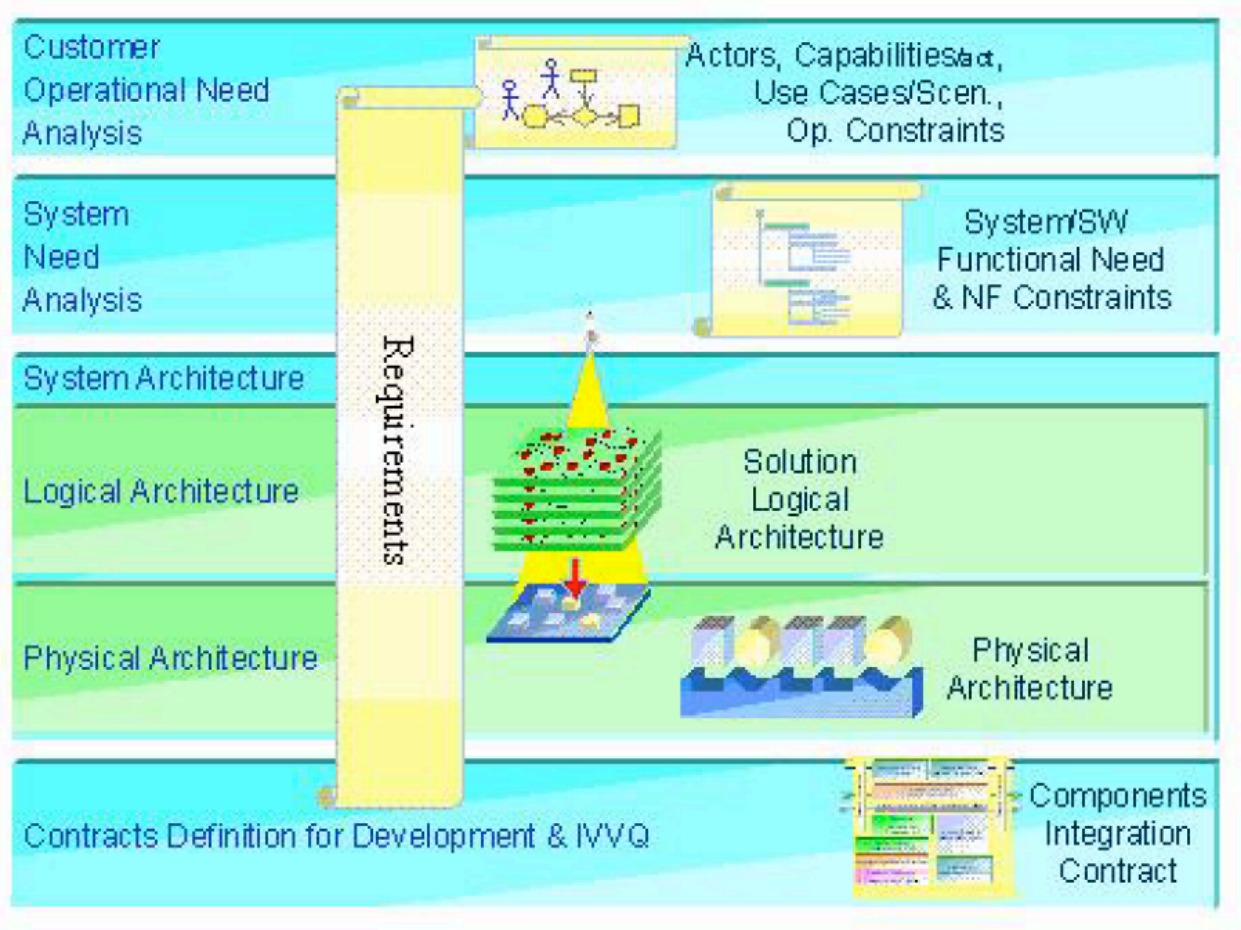
System Need Analysis: This step focuses on defining how the system can satisfy the operational need, along with its expected behaviour and qualities.

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).



ARCADIA Overview

2.3.3 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. In the context of GEMOC we will focused on the Physical architecture view.

The two following figures show the radar model at respectively the operational and physical levels of the ARCADIA methodology.

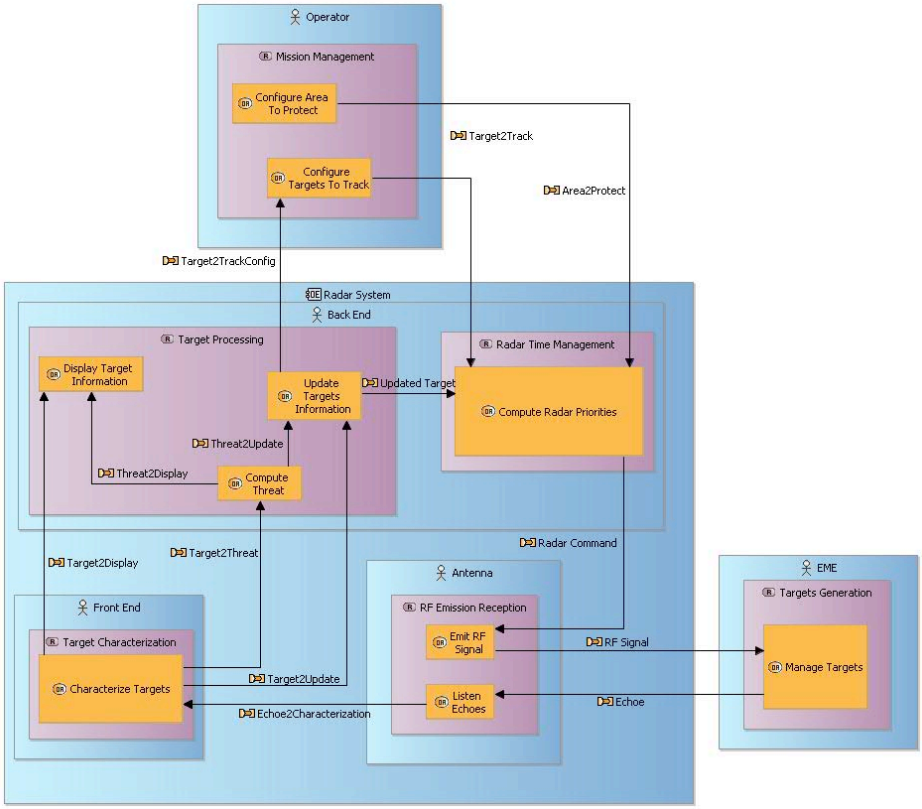


Figure 7: Radar model - Operational Level

At operational level, the Radar system shall manage the targets present in the electromagnetic environment and execute actions based on human / automated decision. The role of the operator is to observe targets displayed on the screen and decide consequently the priorities of actions. He can also use his console to set an area to protect or to choose particular targets requiring more attention. The Radar System has 3 main parts: the antenna sub-system responsible for emitting / receiving Radio Signal to / from the Electromagnetic Environment; the Front-End subsystem, responsible for responsible for transforming signal into exploitable / presentable information (filtering, target characterization, etc.); and finally, the Back-End Sub-system responsible for taking aided / automated decisions.

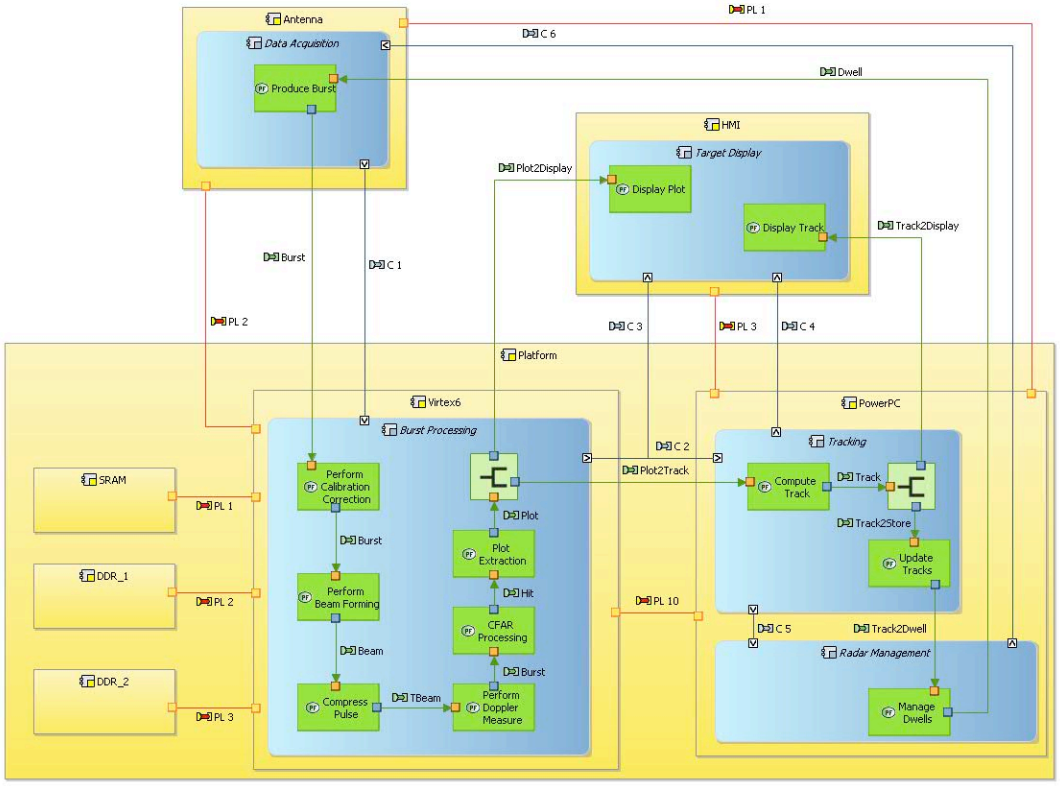


Figure 8: Radar model - Physical Level

At Physical Level, the system is composed of several components. The Burst Processing component running on a Virtex 6 FPGA implements the dataflow functional chain responsible for characterizing the targets located into the Electromagnetic Environment. The tracking component is responsible for assessing the level of criticity of detected target, which will be used later by the Radar Management component to manage the priority of the dwells to send to the antenna. Both Tracking and Radar Management components run on a processor because of the control nature. The Target Display component is used to display qualitative information to the operator. Finally, the Antenna component is responsible for emitting / receiving signals into the Electromagnetic Environment. For the sake of simplicity, the Antenna component in this example is used to simulate the complete Electromagnetic Environment including the targets within it.
We are particularly interested in this example to the combination between two MoCCs: the SDF MoCC implemented on the Virtex6 FPGA and the FSM MoCC implemented on the PowerPC processor.

2.3.4 Behavioural Semantics

The available way to model behaviour 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 behavioural 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 modelling 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 existing features. *Modelling extensions to describe behaviour 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 modelling 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 Physical architecture phase,
2. behaviour modelling 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.

Relevance of objectives for Physical architecture phase	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
Physical architecture	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

3. Requirements 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 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 do- main, i.e. to have a single metamodel shared by different static semantics, e.g., to cope with language variants.

The GEMOC studio shall provide means to assess the correctness of the static semantics at language design time.

The GEMOC studio shall provide means to assess the correctness of the static semantics at language execution time.

Concern #2: Behavioural Semantics Definition

The GEMOC studio shall provide means to define of the behavioural semantics of a DSL. To define the behavioural semantics of a DSL, the GEMOC studio provide the Kermeta Language, an action language that is used to express the behavioural 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. The Model of Concurrency will trigger these methods.

The GEMOC studio shall provide means to assess the correctness of the behavioural semantics at language design time.

The GEMOC studio shall provide means to assess the correctness of the behavioural semantics at language execution time.

Concern #3: Model of concurrency Definition

The GEMOC studio shall provide means to define Model of concurrency either in a declarative (i.e. using constraints "à la" MARTE/CCSL) or an operational manner (i.e. using state machine).

The GEMOC studio shall provide means to connect the Behavioural semantics with the Model of concurrency.

The GEMOC studio shall provide means to assess the correctness of the connection between the MoCC and the other parts of the language at language design time.

The GEMOC studio shall provide means to assess the correctness of the connection between the MoCC and the other parts of the language at language execution time.

Concern #4: Graphical Syntax Definition

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

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

From the end user perspective, Sirius provides:

- Rich and specialized modelling editors to design their models.
- Synchronization between these different editors.

Concern #5: Animation

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

The GEMOC studio shall provide means to execute a model from an xDSML in an interactive or batch manner:

- The batch execution shall take as parameter both the model and input data for the whole execution and produce output data.
- The interactive execution should allow defining breakpoints, running in a step-by-step fashion, or up to the next breakpoint, handling hierarchical execution (in a step over/step into fashion), stopping a run interactively, selecting a DSE from the ones allowed by the MoCC at each execution step.

The GEMOC studio shall provide means to view the evolution of the execution data during a run.

The GEMOC studio shall provide means to store the evolution of the execution Data during a run.

The GEMOC studio shall provide means to store the decision of the user during an interactive run in order to be able to replay it in an interactive or a batch run.

