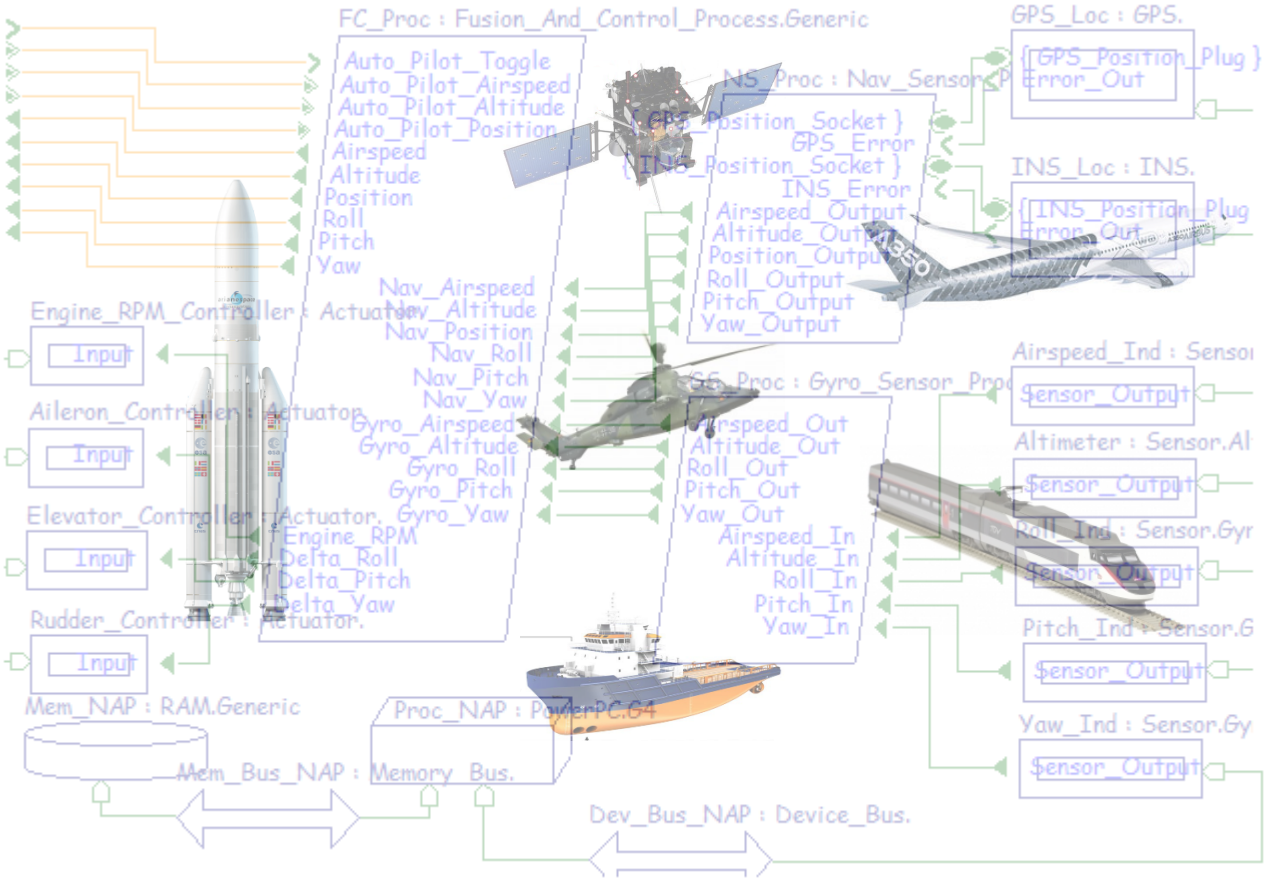
Ellidiss Technologies



Methods and Tools for Mission Critical Software Intensive Systems

任务关键系统&软件开发的方法和工具

Our Activities

Software Tools Editor

Ellidiss Technologies edits tools for the modeling and model processing activities of mission critical software intensive systems development. The company runs three strongly interconnected activities:

- The long-term maintenance and end-user technical support of its HOOD based development tools in operation in major industrial projects, such as Airbus A3xx family, Airbus A400M, Airbus Helicopter Tiger, Eurofighter Typhon, Leonardo AW101, AW149 and M346, as well as various spacecrafts payloads.
- The development of new modeling and model processing tools using the AADL standard either as a native or as a pivot language.
- Contribution to applied research projects and partnership with academic laboratories and industrial stakeholders.

Business

Ellidiss Technologies distributes and supports its products worldwide and has a distributor in China. The company also contributes actively to the SAE AS-2C international standardization committee (AADL) and was the main contractor of an European Space Agency frame contract for the development of the TASTE software engineering tool-set.

Collaborative R&D projects

In order to continuously improve its products, Ellidiss Technologies invests a lot of resources in Research and Development activities by participating in collaborative applied research programs such as IST-FP6 ASSERT, ITEA.SPICES, FRAE QUARTEFT, FUI PARSEC, I&R SMART, IRT St Exupery INGEQUIP, H2020 PERASPERA ERGO and MOSAR. Ellidiss also maintains strong technical relationships with various academic partners, such as Lab-STICC in Brest, Telecom ParisTech, ISAE, LAAS and IRIT in Toulouse and INRIA in Rennes.

taste
The Assert Set of Tools for Engineering

http://taste.tools

ERGO

http://www.h2020-ergo.eu

MOSAR

http://www.h2020-mosar.eu

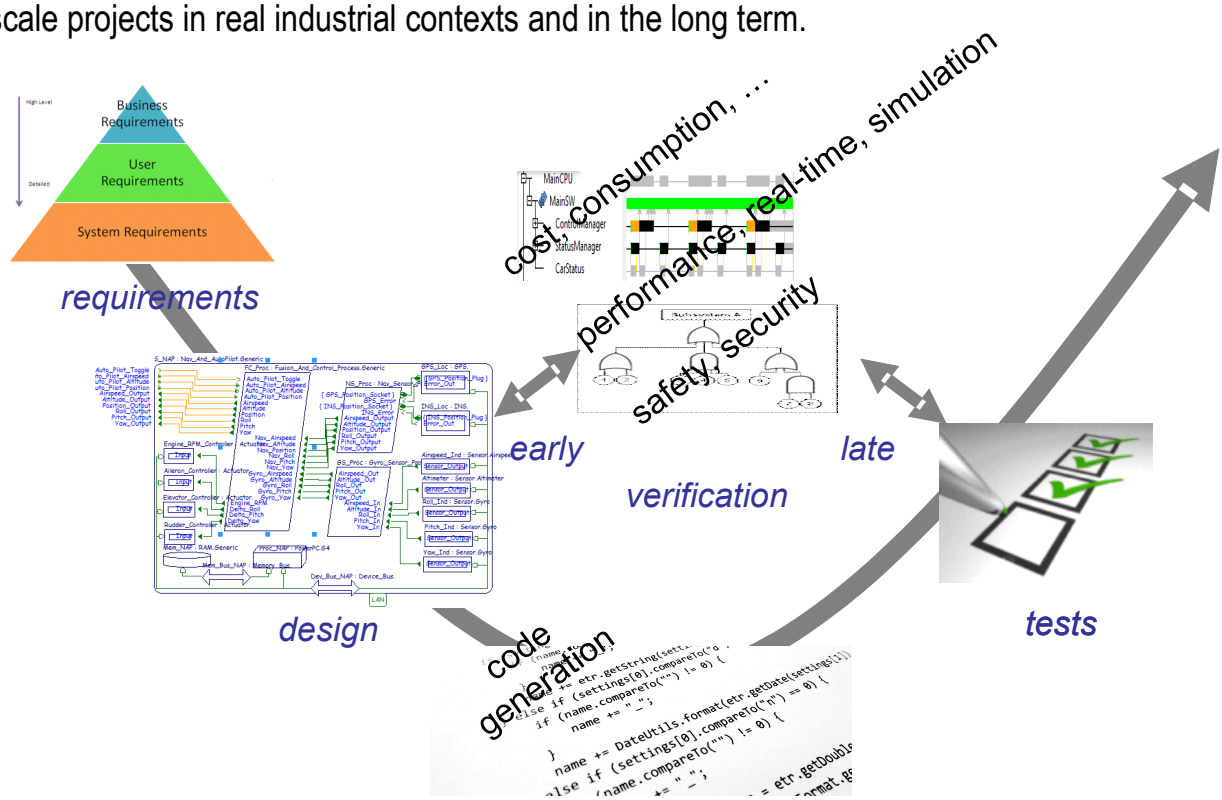
esa
This project has received funding from the European Space Agency under frame contract n° 4000104809

These projects have received funding from the European Union's Horizon 2020 research and innovation programme under grant agreements n° 730086 and 821996

Our Offer

Development of Critical Software:

Due to the increasing importance of Software in critical Systems for many application domains such as Avionics, Space, Ground Transportation, Automotive, Medical equipments, and other Cyber-Physical Systems, Software development teams that are involved in such projects need solutions that help them to:
- Provide a seamless path between System Engineering and Software Engineering activities.
- Support efficiently the various steps of the Software development life-cycle, including Model Driven Engineering, code and documentation generation as well as reverse engineering.
- Offer when appropriate “correctness by construction” modeling approaches, and provide “early verification” solutions otherwise.
- Enforce “good practices” for the development process in order to master the complexity of large scale projects in real industrial contexts and in the long term.



Our solution:

Our solution focuses on the Architectural Design of the future Software application to enable Verification and Production activities that are supported by the following tools and technologies:
- **Stood for AADL**: a graphical editor to directly edit AADL models. It is strengthened by the HOOD industry proven design methodology and advanced project management features.
- **AADL Inspector**: a variety of state-of-the-art independent analysis or production tools efficiently interconnected within a common user-friendly framework.
- **LMP**: an innovative and flexible technology to ensure the required interoperability between the modeling and model processing tools. Dedicated support is provided to process AADL models.
- **GMP**: a flexible toolbox to build domain specific graphical editors.
- **Stood for HOOD** and **CP-HOOD**: mature tools that perfectly fit for long term maintenance of large scale legacy projects in Ada or C.

Our Technologies

Model Driven Engineering

Model Driven Engineering (MDE) solutions enforce the use of modeling languages in the early stages of the system and software development process whereas Code Driven Engineering fully relies on lower level programming languages. The main advantages brought by MDE approaches are that they:
- Reduce the gap between system and software engineering activities.
- Anticipate software testing and integration issues by early model based verifications.
- Increase software development productivity when associated with automatic code and documentation generators.

Modeling languages

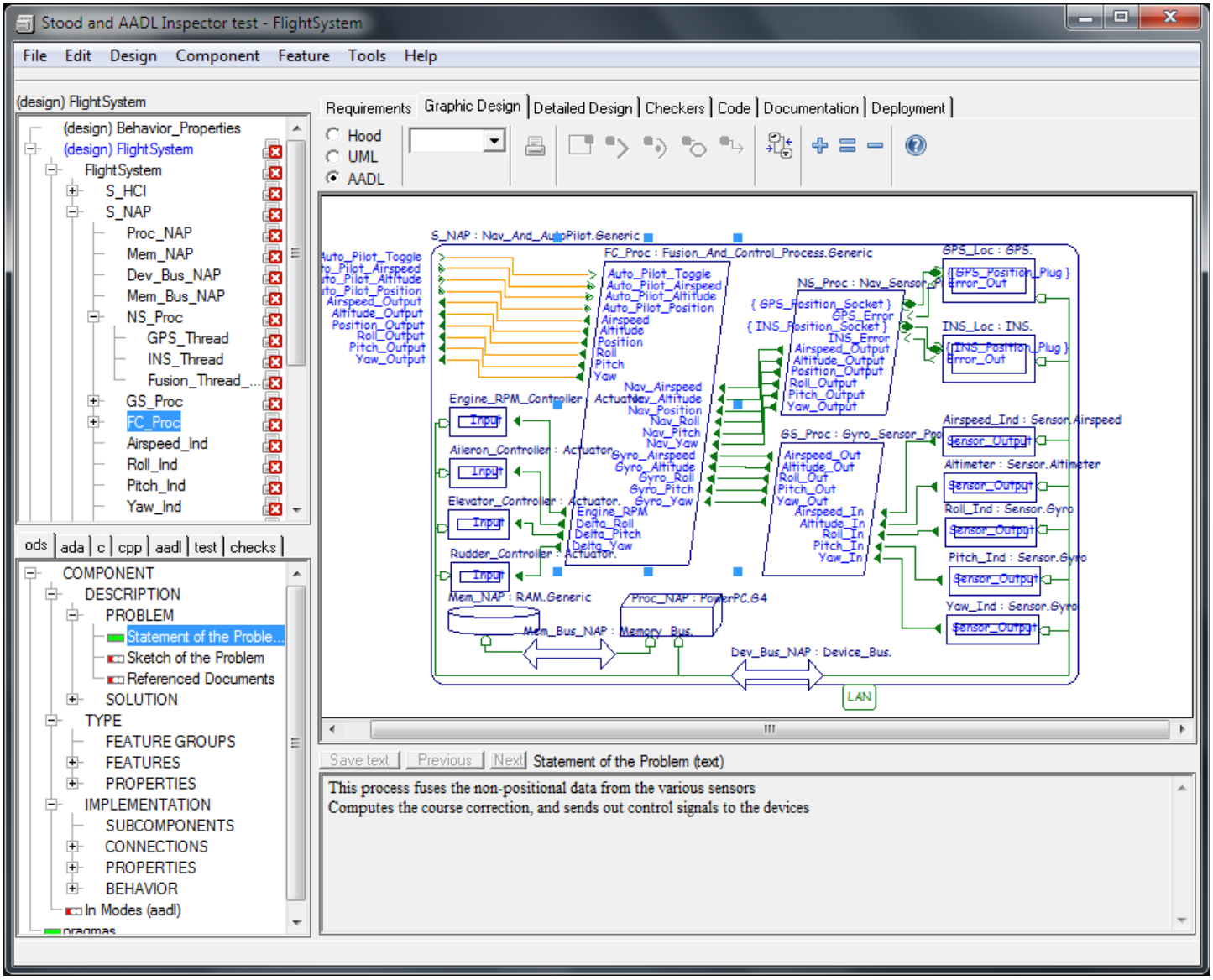
Three main MDE approaches are usually observed:
- Direct use of OMG standards such as UML, that must be enriched with one or more profiles (i.e. SysML, MARTE) to carry out advanced model processing tasks.
- The specification of a Domain Specific Modeling Language (DSML) expressed by a proper meta-model and requiring the development of appropriate tools (i.e. Capella).
- The use of a standard Architecture Description Language such as AADL to define a common infrastructure of the system that can be completed by domain specific properties or sub-languages to cover the various model processing requirements.

Architecture Analysis and Design Language

Without rejecting the other approaches, we have selected the Architecture Analysis and Design Language (AADL) to be used either as a native or as a pivot modeling language, as it is the only one that offers:
- A high level of semantics that enables the use of advanced model processing for performance, safety and security analysis without custom extensions.
- A set of pre-defined modeling constructs that are as close as possible to the engineering domain, without being dedicated to a too small segment of the development life-cycle.
- An architecture oriented approach that provides modularity and scalability to manage the increasing size and complexity of real industrial systems.
- A textual representation that ensures models accessibility as well as easy configuration and version management.

Stood for AADL

Although the AADL is a textual language, the standard also defines a graphical notation that can be used to illustrate hierarchical architectures. The Stood tool, developed by Ellidiss provides much more than graphical editing. It supports a complete architectural and detailed design process based on the HOOD (Hierarchical Object Oriented Design) methodology starting from import of software requirements to code and documentation generation.



Declarative and Instance Models

Like all Object-Oriented languages, the AADL declarative model specifies a set of classifiers. This organization is well appropriate for defining libraries of reusable components. However, in most cases, advanced model processing requires access to a completely expanded instance tree, starting from the root system down to the low level elementary components and taking into accounts the software distribution on the hardware. This intuitive view of the system can be directly edited graphically with Stood and processed by AADL Inspector.

AADL

The Architecture Analysis and Design Language (AADL) has been standardized by the SAE, avionics systems division, under number AS 5506. Current version of the standard (v2.2) has been issued in 2016. Version 3 of the standard is in preparation. The AADL is a textual and graphical language that can be used to design and analyze the software and hardware architecture of performance-critical real-time systems. An AADL model describes a system as a hierarchy of components with their interfaces and connections.

AADL components

AADL components belong to predefined categories for which precise semantics are defined by the standard.
- Execution platform components represent the hardware parts of the system which may be: Processors, Buses, Memories and Devices.
- Application software components describe the applicative model running on the system in terms of: Processes, Threads, Subprograms and Data.

AADL components have a type that describes their functional interface and one or several implementations defining their internal structure and in particular their sub-components and sub-components interactions.

AADL features and connections

The purpose of AADL features is to specify the functional interface of a component. A feature can be an incoming or outgoing port (event, data or event data), a subprogram entry point or a data access point. Features are ends for AADL connections.

AADL properties

Default semantics of AADL constructs are described by the standard, and appear in the models either implicitly (run-time semantics) or explicitly with predefined properties. Additionally, it is possible to enrich the semantics of AADL models by defining user-defined property sets that can apply to AADL component categories, features and connections.

SAE International
http://standards.sae.org/

AADL Extensions

The AADL standard comes with a set of extensions that enlarge the scope of the language for more particular usages. These extension capabilities include the definition of new Properties to annotate the architecture (Property Sets) and sub-languages (Annexes). Some of these Annexes have already been published and others are still being studied by the AADL committee.

- Error Model Annex.
- Behavior Annex.
- Data Modeling Annex.
- Code Generation Annex.
- ARINC 653 Annex.
- Security Annex.

Verifiable Model Driven Engineering

One of the main benefits of using the AADL for modeling critical systems is that this language has been developed in the purpose of being verified. The standard specifies numerous legality rules that can be checked automatically by tools to insure that the model is correctly formed according to the AADL semantics. Moreover, a common architectural description of the system can carry a variety of information that may be processed separately in order to provide analysis reports in various areas:

- Static Analysis (Modeling Rules, Footprint Analysis, ...)
- Timing Analysis (Schedulability, Simulation, End to End Flows Analysis, ...)
- Safety Analysis (Fault Tree Analysis, Failure Modes and Effects Analysis, …)
- Security Analysis (Confidentiality, Integrity and Availiability Analysis, ...)
- Model exploration and inline processing with the LAMP verification language.

AADL tools

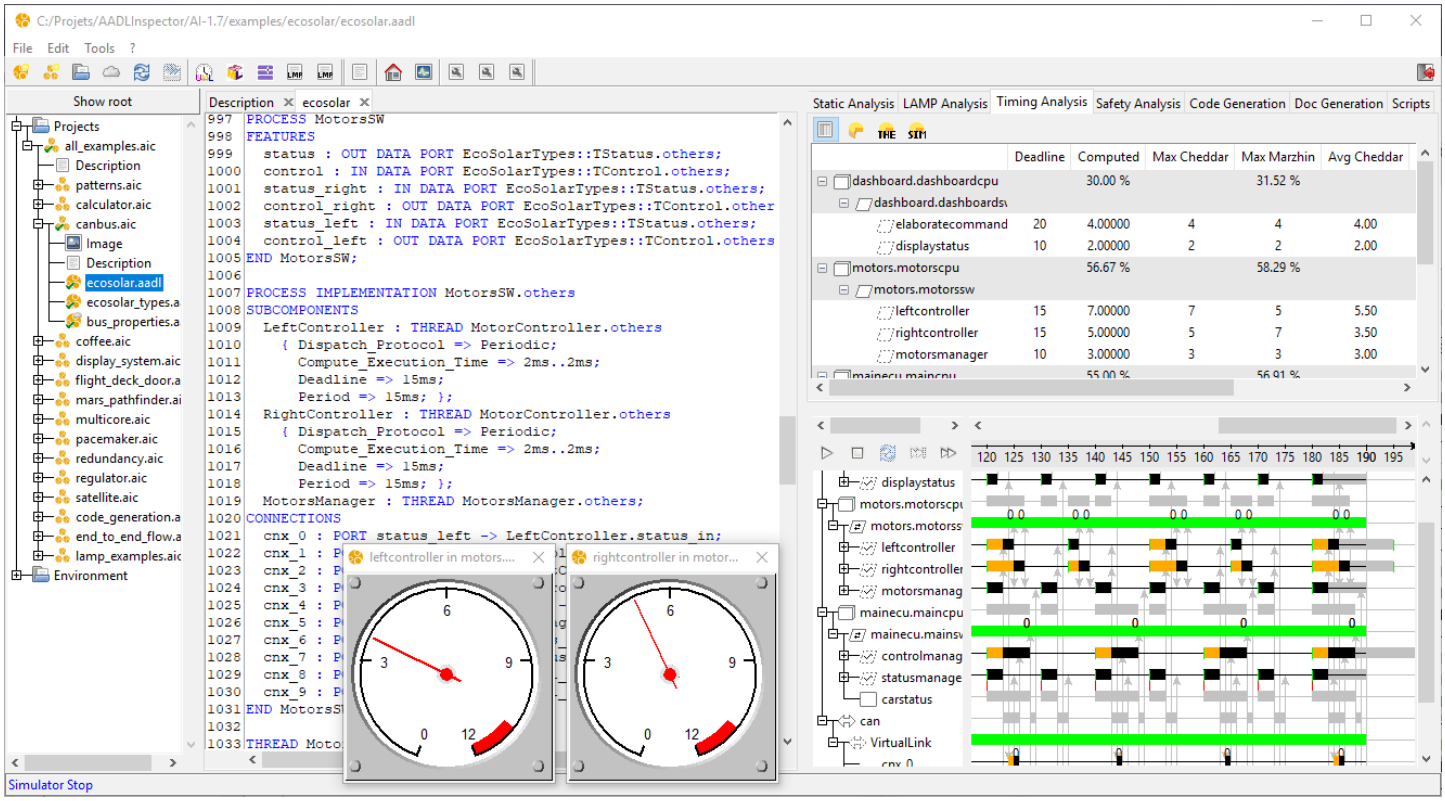
Since the first release of AADL in 2004, many modeling and verification tools have been developed. Several Open Source tools have been developed by Universities or Research Institutes. Ellidiss and its partners develop and distribute commercially supported AADL tools and has become a leader in this area. These products are:

- **STOOD for AADL**: graphical editor, code and documentation generation, requirements traceability.
- **AADL Inspector**: modular analysis framework for AADL. It includes a set of static analysis tools, the **Cheddar** schedulability analysis tool, the **Marzhin** dynamic simulator, the **Ocarina** code generator and full access to model exploration and inline processing thanks to the embedded **LAMP** verification engine.
- **TASTE**: complete AADL based tool-chain developed for the European Space Agency.

In a more general point of view, the use of AADL as a pivot language enables the realization of complete cost-effective tool chains that can be customized to fit the constraints of industrial software development processes.

AADL Inspector

AADL Inspector is a light and modular framework for processing AADL models. It makes an intensive use of the LMP technology. This product can read AADL specifications and apply to them a variety of processing tools organized as modular plug-ins and managed separately. Each processing plug-in is declared by a configuration file that specifies whether a model transformation is required or not, and if a remote tool has to be called. AADL Inspector is the ideal way to connect specialized tools that are being developed independently, without requiring them to be written in a unique language or already integrated within the same framework. In other words, AADL Inspector does not require all the processing tools to be implemented in EMF/Java. AADL Inspector also enriches the AADL standard by defining hierarchical project structures, model execution scenarios and inline LAMP verification rules.



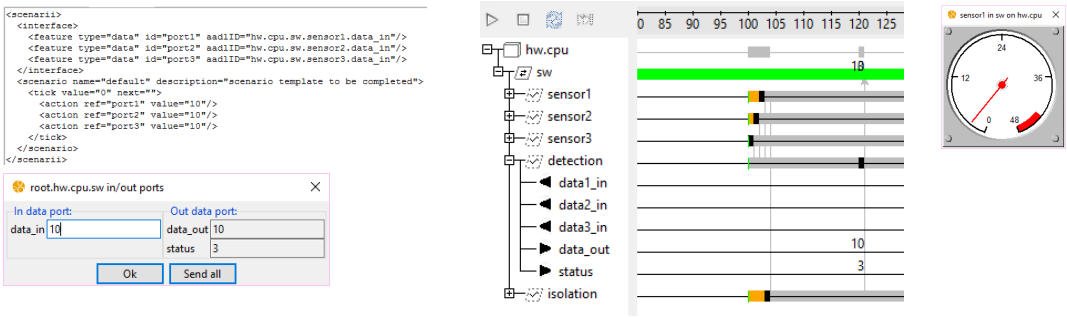
AADL Inspector plug-ins

AADL Inspector plug-ins can be added in a “plug and play” manner. However, the product comes with a set of already installed tools:

- Import of UML/MARTE, SysML, Capella or FACE models.
- Static rules analyzers, for compliance checking with AADL legality rules.
- Scheduling analysis and static simulation over the hyper period with Cheddar.
- Interactive event based simulation with Marzhin.
- Scheduling Aware end to end Flow Latency Analysis (SAFLA).
- Fault Tree Analysis with a connection to Arbre Analyste as an external tool.
- A LAMP interpreter to let the user dynamically explore and process the AADL model.
- A set of Security verification rules templates implemented with LAMP.
- Software source code generation with Ocarina.
- Analysis report production with an embedded PDF document generator.

Virtual Execution of the model

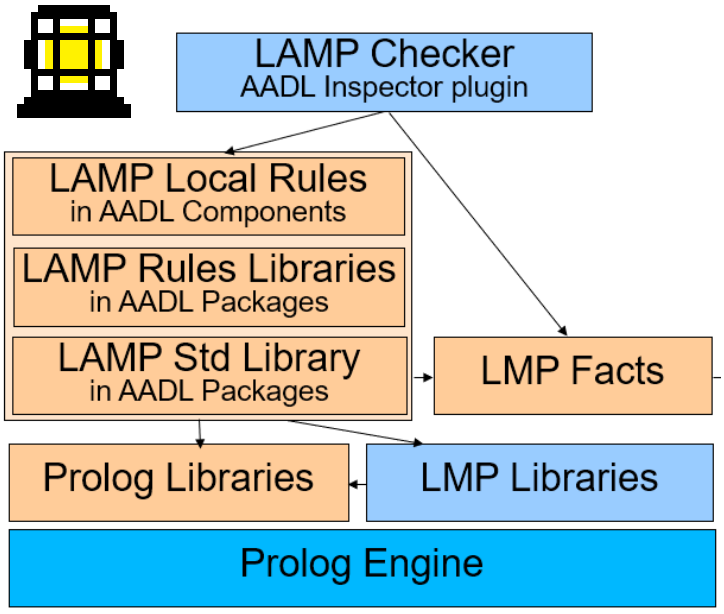
AADL architectures can be enriched with standard Behavior Annexes to express the functional behavior of the future system. Behavior Annexes are fully compliant with the run-time semantic defined by the core of the AADL standard and define finite states automata that can typically be attached to thread and subprogram components. Each transition may lead to the execution of a sequence of actions, including reading and writing port values, sending events, calling subprograms, defining critical sections and performing arithmetic operations. The Marzhin simulator can then interpret these statements and virtually executes the AADL model for the purpose of an early verification of the desired behavior of the system to design. The AADL simulation can interact with the user thanks to automatically generated dialogues or with pre-defined scenario files that can be stored together with the AADL model in the project structure.



Logic AADL Model Processing (LAMP)

LAMP is an original and powerful model processing language that can be directly embedded within AADL models as a textual annex. It is based on the use of the Prolog language and the LMP framework. LAMP is especially useful for:

- Rapid prototyping of new model processing plugins.
- Development of customized (private) processing tools at corporate or project level.
- Model exploration and debugging.



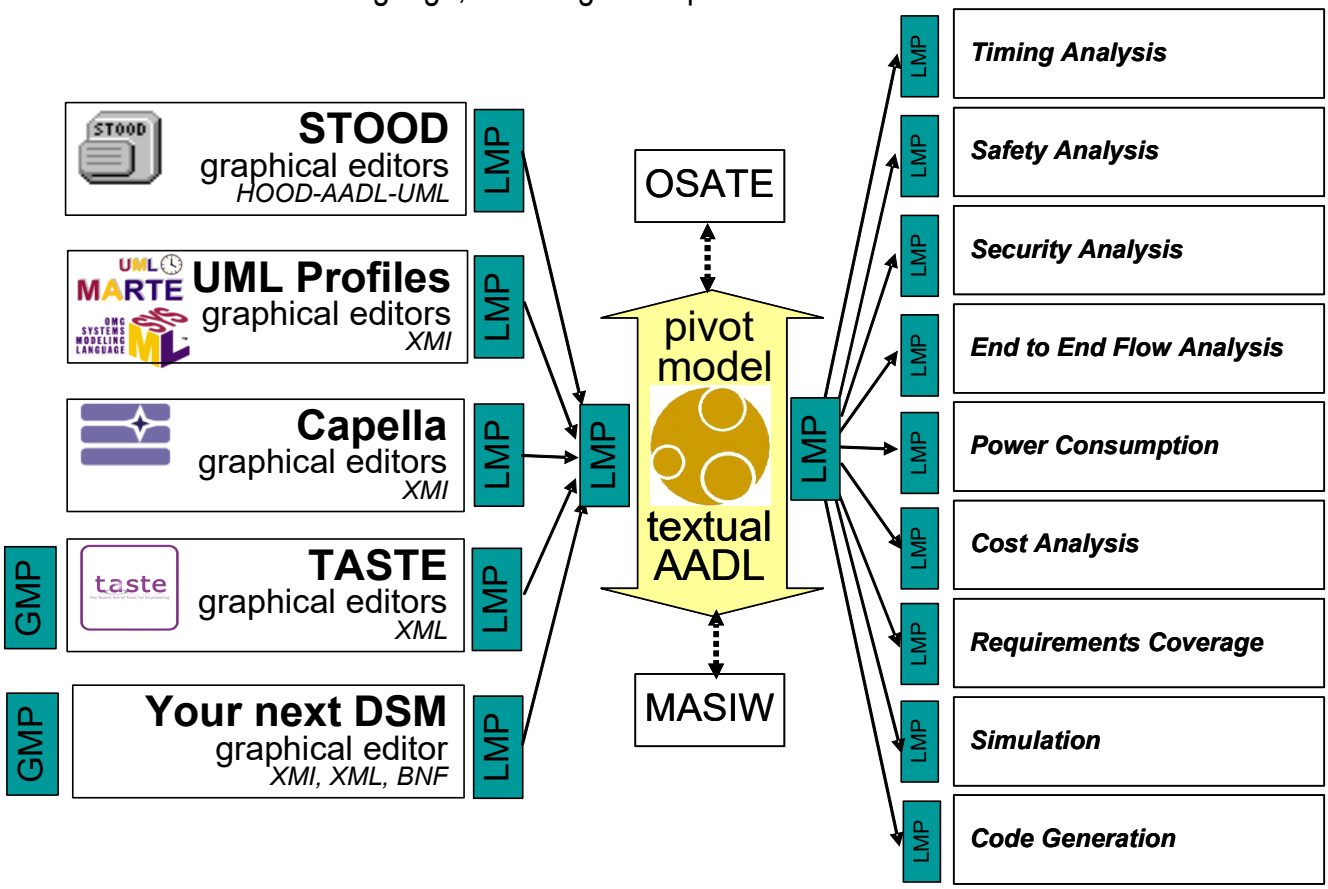
Model Processing

Although many efforts are still required in order to improve the way large scale and complex models are edited, today's challenge in Model Driven Engineering mostly consists in how these models can be processed. By model processing, we mean:
- Model Queries.
- Model Constraints.
- Model Verifications.
- Model to Model Transformations.
- Model to Text: Source Code and Documentation Generation.

All these needs can be satisfied by the LMP technology that can be used to develop tools plugins (offline) as well as by the LAMP dynamic checker that enables user defined processing rules inside the model to be analysed (inline). Both use the power of the Prolog language at model level.

Tool Chains

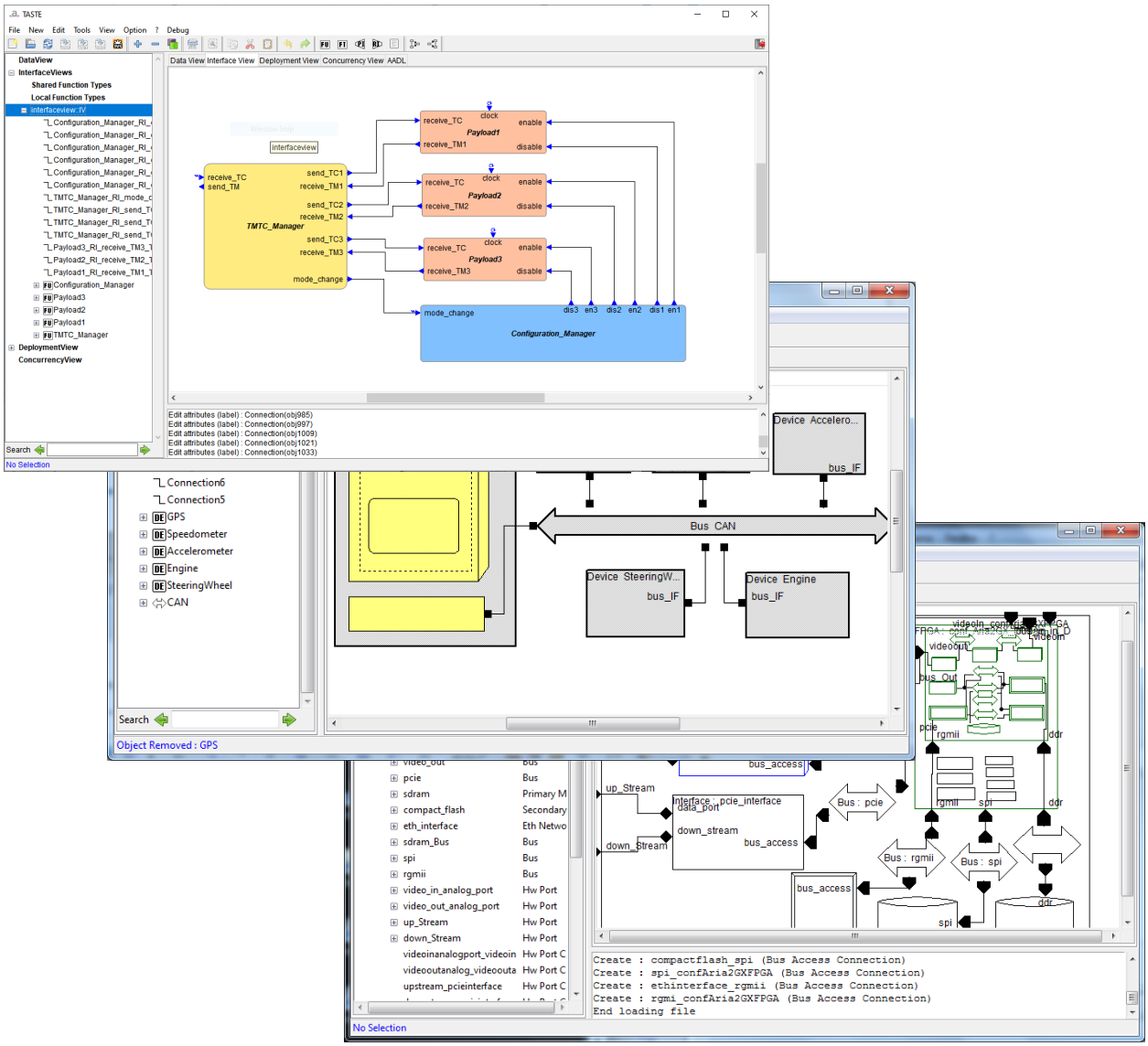
Due to the variety of modeling languages and tools and the increasing need for advanced model processing modules, Ellidiss Technologies has defined a modular and flexible solution to ease the development of complete tool chains. This approach is based on three main technological components:
- The AADL language, that can be used as a native modeling language or as a hidden pivot language when a UML profile or a DSML is used at the front end.
- The GMP technology to develop Domain Specific graphical front ends that produces standard AADL models.
- The LMP technology that is used to implement all the model processing needs with a common standardized language, including the import of XML or XMI based models.



GMP: Graphic Model Processing

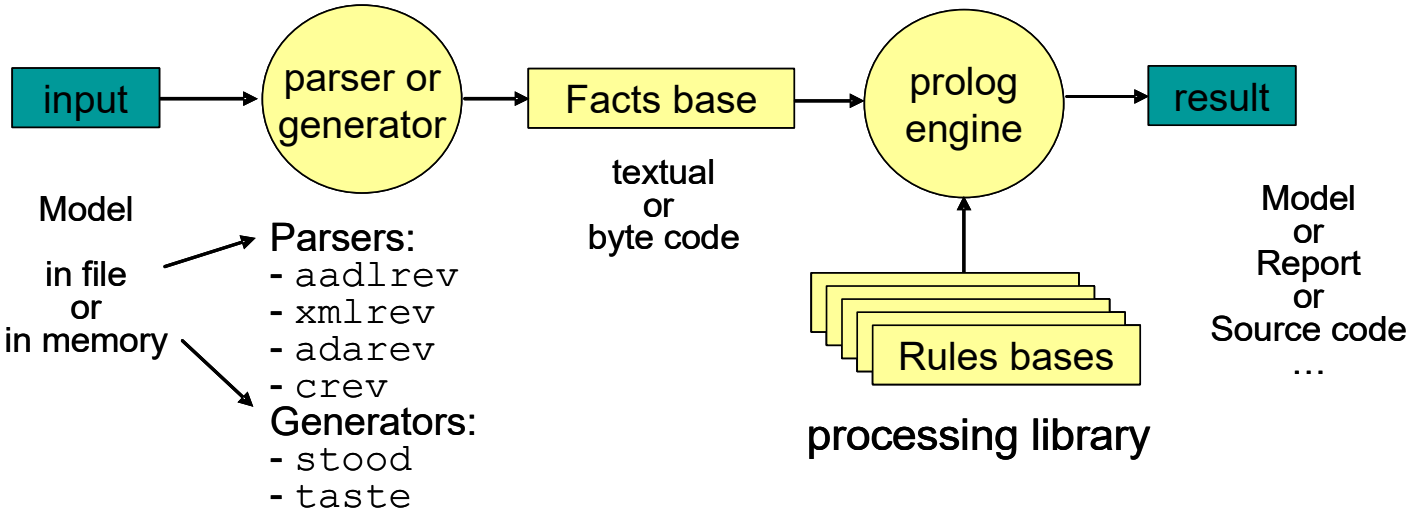
AADL perfectly fits the need for engineering teams who want to fully master the details of their real-time embedded system architecture with a standardized notation. However, in many cases, system architects prefer to use a Domain Specific textual or graphical Modeling Language (DSML) in order to work with custom concepts or maintain legacy projects. Ellidiss Technologies offers a seamless way to take profit of both a specific approach for modeling activities and generic tools for model processing. This combined solution requires to execute the following steps:

- Formal identification of the used domain specific language (meta-model).
- Development of a light graphical tool, using the Graphic Model Processing (GMP) technology.
- Specification of a semantic mapping between this DSML and standard AADL constructs.
- Implementation of automatic model transformation from and to AADL, using the Logic Model Processing (LMP) technology.



LMP: Logic Model Processing

LMP applies the principles of Logic Programming to Model Driven Engineering. In a few words, each model is expressed by a list of prolog facts and each processing function is implemented as a list of prolog rules. Applying the rules to the predicates will produce the awaited result (verification report, target model, source code or documentation text).



LMP benefits

- LMP offers a single solution to implement model queries, model constraints and model transformations instead of having to use several dedicated languages
- LMP is based on the standard prolog language (ISO/IEC 13211-1)
- LMP is declarative, modular and formal (boolean logic), which open the door for tool qualification.
- LMP is flexible and can be used to process heterogeneous models or incomplete models.
- LMP is commercially supported and benefits from industrial return of experience
    - **Airbus**: involved in the development of DO-178 certified projects (A380, A350)
    - **European Space Agency**: used in the TASTE graphical editors
    - **Ellidiss**: AADL Inspector model adaptors and Stood code generators

LMP Dev Kit

The LMP Dev Kit provides the LMP designer with a set of tools and libraries to increase the productivity and the quality of the development of LMP applications. This toolbox is composed of three compartments:
- Tools for the generation of Facts bases (AADL and XML/XMI parsers)
- Tools for the development of Rules bases (Ecore2LMP and LMP Designer graphical editor)
- Tools for the run-time execution of the LMP application (sbprolog)

Analysis Use Cases

Real-Time Systems

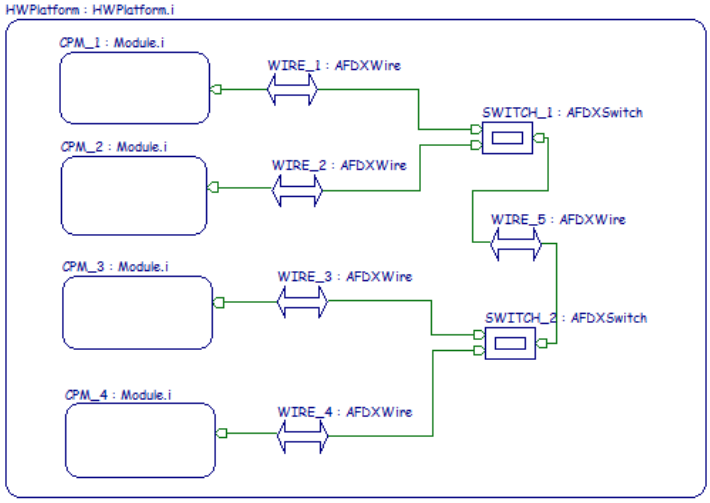
Modeling and early validation of real-time features is a key concern for embedded applications. With our modeling and verification approach, it is possible to check that the task set is schedulable, that there are no deadlock or priority inversion. The most popular scheduling protocols are supported: Rate Monotonic, Deadline Monotonic, POSIX, Earliest Deadline First, …

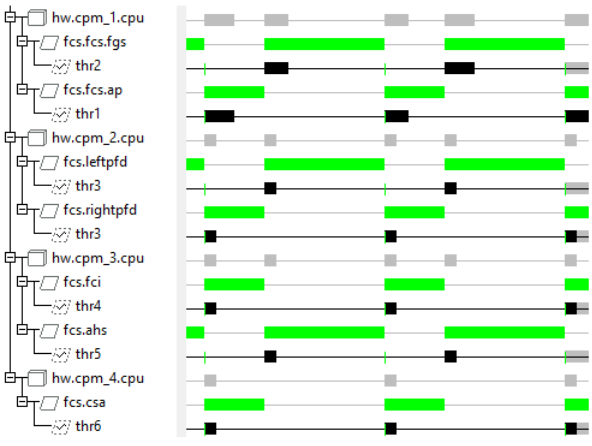
The response time of each thread as well as the processor load are estimated according to three different computation methods: Theoritical feasibility tests and static simulation over the hyper-period that are directly provided by Cheddar, whereas statistical values are computed from the output of the Marzhin event based simulator. All these results are summarized in a table for an easier analysis.

	Deadline	Computed	Max Cheddar	Max Marzhin	Avg Cheddar	Avg Marzhin	Min Cheddar	Min Marzhin
rs_6000		72.50 %		82.46 %				
prs_psc								
bus_scheduling	10	2.00000	2	2	2.00	2.00	2	2
data_distribution	10	4.00000	8	8	4.10	4.33	4	4
control_task	20	6.00000	10	10	6.20	6.67	6	6
radio_task	20	8.00000	16	16	8.40	9.33	8	8
camera_task	20	10.00000	18	18	10.40	11.33	10	10
mesure_task	400	18.00000	18	18	18.00	18.00	18	18
meteo_task	400	38.00000	26	26	26.00	26.00	26	26

Time and Space Partitioning

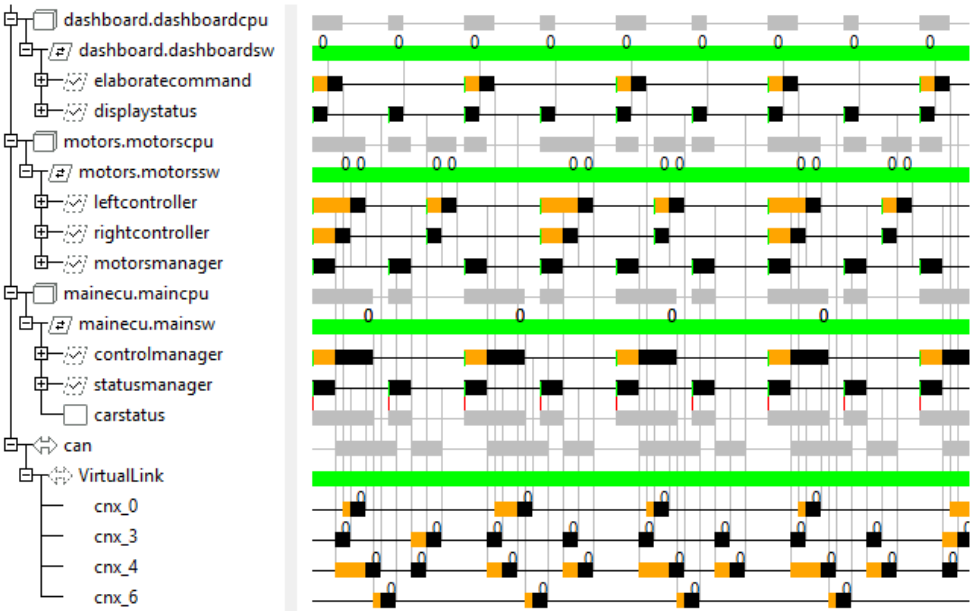
Time and Space Partitioning (TSP) is a required practice for Integrated Modular Avionics (IMA) or security management. Such architectures can be modeled and analyzed thanks to the use of the ARINC 653 AADL Annex. The Marzhin simulator can emulate the scheduling of the partitions on the processor, and of each task set on each partition.





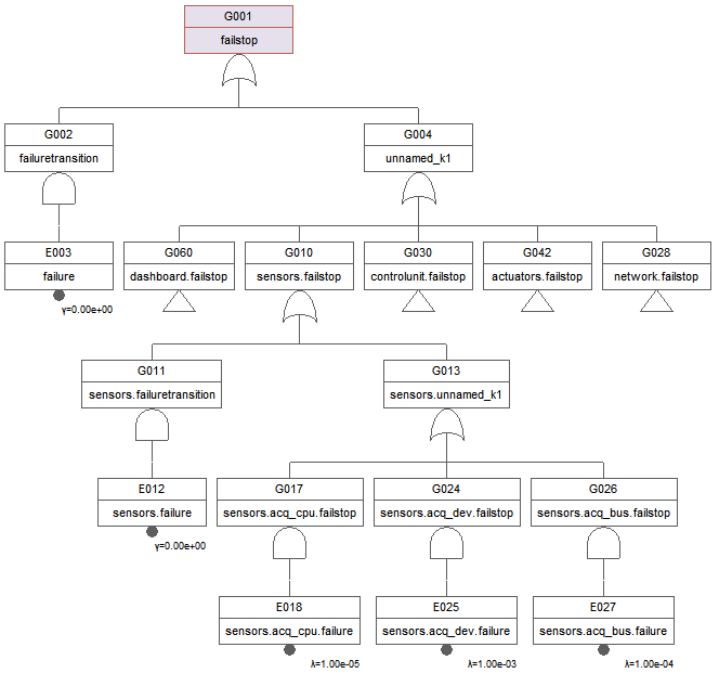
Global network timing analysis

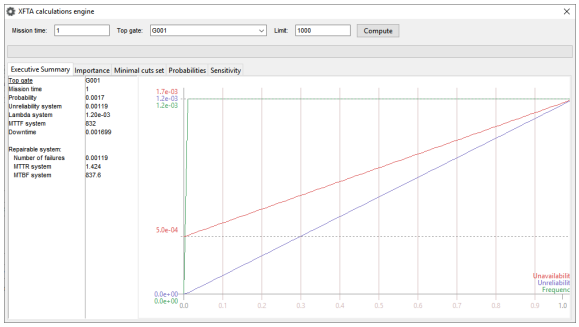
This approach fully supports the modeling and global timing analysis of several processors connected by a bus. Each node of the network is described by a Real-Time software running on a processor and each inter-node communication generate messages that are schedules on the bus.



Safety Analysis

Interpretation of Error Model annexes included inside AADL architecture models can be used to perform usual safety analysis activities. AADL Inspector implements an AADL to OpenPSA model transformation and a connection to the remote Arbre Analyste tool to perform Fault Tree Analysis:





http://www.arbre-analyste.fr/en.html/

HOOD

The Hierarchical Object Oriented Design (HOOD) method was created in the late eighties by the European Space Agency, in order to provide a Model Driven solution on top of the Ada programming language for its large scale projects such as Ariane, Hermes and Columbus. It has been applied to the most important avionics and space projects in Europe and continuously supported by Ellidiss.

As opposed to the AADL or UML, HOOD is not only a modeling language. It also defines a complete development process from requirements analysis to Ada, C or C++ code generation of distributed real-time Software.

Mastering complexity

With HOOD, a project is composed of a set of main modeling units called Designs. A design can lead either to a completely autonomous Software application (an executable file) or a library of linkable Software entities (functions, data types, classes, ...).

Each HOOD design is the root of a hierarchy of components that is usually built following a top-down approach that drives the architectural design phase of the Software development process.

Preventing “spaghetti”wares

Each HOOD component can be represented by a black box view that exposes a Provided Interface and a Required Interface, and a white box view that shows sub-components in case of a non-terminal object of the design hierarchy or the implementation of the actual Software entities (functions, data types, variables, constants, exceptions) in case of a terminal object.

The criteria for this top-down decomposition of the System is given by the very intuitive « low residual coupling » principle: all the entities that interact the most together must be located within the same branch of the hierarchy. This leads to an architecture where residual connections between the various Software units are minimized, which is a key for building easy to integrate and to maintain Software applications.

HOOD and AADL

Due to the numerous similarities between the AADL Software components and the architectural design constructs offered by HOOD, Ellidiss Technologies has been able to integrate most of the AADL into its Stood tool which was initially developed to support the HOOD methodology.

By offering both AADL and HOOD inside its Stood tool, Ellidiss provides a unique solution to help AADL users to master the complexity of large software architectural models.

Edited By :
HOOD User's Group

Ellidiss Technologies

30 years of support for major industrial projects

HOOD design tools for Ada and C/C++:
- *Eurofighter Typhoon*.
- *Airbus A340, A380, A350, A400M*.
- *Tiger helicopter.*
- *Rafale fighter.*
- *Leonardo AW101, AW149 helicopters and M346 trainer*
- *Legacy Ada code reverse engineering*.

15 years of investment in innovative software technology

- Contribution to the AADL standardization committee (SAE AS-2C).
- AADL graphical modeling tools: Stood for AADL, Adele, TASTE.
- Model processing runtime framework: AADL Inspector.
- Model processing development framework: LAMP, LMP Dev Kit.
- Contribution to collaborative R&D projets

Products and Services

- Commercial Off-The-Shelf (COTS) software tools: **CP-Hood, Stood**, **AADL Inspector**.
- Technical support, Training, Tool customization, Maintenance.
- Technology: GMP (graphical editors) and **LMP** (model processing).
- Specific software tool development.
- Tool prototype development for collaborative research projects.

Ellidiss Technologies
www.ellidiss.com
info@ellidiss.com
24 quai de la douane
29200 Brest
France
+33 (0) 298 45 18 70