TN
UML2 for System Modeling

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Signature</strong></td>
</tr>
<tr>
<td>Prepared by:</td>
<td>Silvia Mazzini</td>
</tr>
<tr>
<td>Approved by:</td>
<td>Silvia Mazzini</td>
</tr>
<tr>
<td>Authorized by:</td>
<td>Dario Citterico</td>
</tr>
<tr>
<td>Issue</td>
<td>Revision</td>
</tr>
<tr>
<td>-------</td>
<td>----------</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENT

1 **INTRODUCTION**
- 1.1 Scope
- 1.2 Background
- 1.3 Purpose
- 1.4 Applicable Documents
- 1.5 Reference Documents
- 1.6 Bibliography Update
- 1.7 Acronyms
- 1.8 Glossary
- 1.9 Study Logic

2 **SPACE SYSTEM ENGINEERING REQUIREMENTS**
- 2.1 Introduction
- 2.2 Space System Engineering
- 2.3 Space System Processes and relevant Phases
- 2.4 Space System Features
- 2.5 Design Properties
- 2.6 Other Language Features

3 **UML AND RELEVANT PROFILES**

4 **SURVEY OF PROFILES**
- 4.1 AADL
  - 4.1.1 What is
  - 4.1.2 Synopsis
  - 4.1.3 Tools
- 4.2 SysML
4.2.1 Introduction 22
4.2.2 SysML Design Principles 24
4.2.3 SysML Diagrams 25
4.2.4 Structure 26
4.2.4.1 Standard Ports ................................................................. 28
4.2.4.2 Flow Ports ........................................................................ .28
4.2.5 Cross-Cutting Constructs 29
4.2.6 Requirements 29
4.2.7 Allocations 32
4.2.8 Parametric Diagrams 33
4.2.9 Behavior 35
4.2.10 Continuous Systems 36
4.2.11 Probabilities 37
4.2.12 Tools 37
4.3 MARTE 37
4.3.1 Introduction 37
4.3.2 The Approach 38
4.3.3 Modelling RTE Systems 38
4.4 SoC 38
4.5 System C 39
4.6 OMEGA 40
4.7 TUT 41

5 ANALYSIS OF PROFILES 44
5.1 Introduction 44
5.2 Relationships among profiles 44
5.3 Allocation of Profiles to the software life cycle 44
6 SYSML COMPLIANCE TO SYSTEM ENGINEERING REQUIREMENTS

7 USING SYSML FOR SPACE SYSTEM AND SOFTWARE ENGINEERING
   7.1 Using SysML for Space System Engineering
   7.2 A Model for the System Engineering Processes
   7.3 Using SysML for Software Related System Engineering
   7.4 Moving from System Engineering to Software Engineering

8 REQUIREMENTS FOR SYSML TOOL SUPPORT
   8.1 General Requirements
   8.2 Support for Requirement Modeling
   8.3 Support for Block Diagrams

9 CONCLUSIONS
1 Introduction

1.1 Scope

This document is a deliverable of the ESA project “Support activities for TEC-SWE and TEC-SWM Sections”. The project is identified as the CCN 02 to ESA/ESTEC/Contract No.17764/03/NL/JA.

1.2 Background

The new emerging UML2 OMG standard enables the possibility to provide standard UML2 unified support for the modeling of the space system and software, across all the system and software engineering processes from specification, architecture and design, to implementation, verification and validation.

Space system and software models can be related by the adoption of new and existing UML2 profiles, such as SysML [R.1], AADL [R.2], or HRT-UML2 [R.3].

SysML, which is currently in the process of OMG standard specification, is a customization of UML2 for systems engineering, to support modeling of systems, which may include hardware, software, data, personnel, procedures, and facilities, focusing on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem.

The Avionics Architecture Description Language (AADL) is an architecture description language (ADL), aimed to embedded real-time systems. The Society of Automotive Engineers (SAE) is developing a standard for AADL, using UML and the Honeywell's MetaH architecture language as a base, and Eclipse tools. The AADL is designed to support the specification and analysis of predictable, performance-critical systems including real-time, fault-tolerant, safety-critical, securely partitioned, dynamically reconfigurable multi-processor systems, and system-of-systems architectures. AADL is the existing language for systems modeling of choice in the ASSERT EC 6th Framework Embedded Systems IP Project.

The Hard Real-Time-Unified Modeling Language (HRT-UML) is a language and method for the design of space software real time systems. The definition of the HRT-UML2 Profile [R.5] finalizes the HRT-UML method definition by implementing the outcomes of the ESTEC validation study and mapping a number relevant concepts to semantically consistent elements of the UML2 meta-model, while the notions of the UML2 meta-model contribute to clarify the methodological use of the corresponding HRT-UML concepts.

In the frame of ASSERT, the HRT-UML/RCM methodology is currently developed, supporting the development of reusable space real-time on-board software. HRT-UML/RCM is based on profiling of the UML2 OMG standard and adopts a model-driven approach for the development of software compliant to the Ravenscar Computational Model (RCM) for predictable concurrent real-time computing. HRT-UML/RCM includes integrated capabilities for:

- Feasibility analysis for timing and sizing
1.3 Purpose

The purpose of the deliverable is to assess the utilization of UML languages for system requirement capture and system design. This concerns the assessment of existing UML profiles for system engineering, and their positioning with respect to the space system and software life cycles and with respect to other system and software languages and methodologies for requirement analysis and design.

In the context of space projects, it is required to analyze the capabilities of system engineering languages and methodologies to address in particular the following aspects:

- Capability to describe formally the link between a functional system architecture to a physical / topological architecture. The physical domain encompasses mechanical, electrical and logical (SW) elements. (This basically addresses the step from the user requirements to the technical specification.)

- Capability to describe the interaction between the different domains, especially the capability to “propagate” changes occurring in one domain into another one.

If these areas can be successfully addressed with SysML, the link between these descriptions on system level to the domain specific languages shall be outlined, and any limitations highlighted.

The assessment of dedicated UML profiles to support embedded systems design, such as, for example UML for System on Chip, MARTE proposal, and the UML for System C are also performed.

An outline of the languages/methodologies using this formalism is produced, covering all the steps of the life cycle (i.e. from requirements to qualification).

Based on the insight gained in the analyses performed, a short overview of available tool support for the formalism is performed and an evaluation of the suitability as support for the outlined methodology is made.

1.4 Applicable Documents

[A.1] ESTEC SOW, Appendix 1 to CCN 02 to ESA/ESTEC/Contract No.17764/03/NL/JA, April 2006.


1.5 Reference Documents


1.6 Bibliography Update

Note that OMG technology adoption of standard specifications, after the initial and revised submissions, foresees an update iterative process, including the following phases:

- A first revision process, via a Finalization Task Force (FTF), for initial tuning and bug fixing of the standard,

- A maintenance process, via a Revision Task Force (RTF), collecting and acting on submitted issues, for the production of revised specifications.
This leads to a continuous update of standard documents, so that their version may change in the time frame of this project duration.

This study is based on applicable and reference specifications, as they were available on July 2006, when this activity started. Their versions are duly specified.

### 1.7 Acronyms

- **AADL** Avionics Architecture Description Language
- **HRT** Hard Real-Time
- **UML** Unified Modeling Language
- **HOOD** Hierarchical Object Oriented Design
- **ESA** European Space Agency
- **ESTEC** European Space Agency Technical Centre
- **GUI** Graphical User Interface
- **OBCS** Object Control Structure
- **ODS** Object Description Skeleton
- **OOD** Object Oriented Design
- **SDL** Specification Description Language
- **SW** Software
- **XMI** XML Metadata Interchange
- **XML** Exchange Markup Language

### 1.8 Glossary

**Class**

A class is a model element describing the structure and behavior of objects. In the context of this study, we are mainly interested on discussion about classes and class instances, and we use the class term in the object oriented classical sense, but also including the more general **UML classifiers**.

**Classifier**

A UML classifier is a model element that describes things (objects) having identity, state, behavior, relationships and an optional internal structure. There are several kind of elements in UML, such as interfaces, data types, use cases and signals, that collectively are called classifiers,
and they behave much like classes, with some additions and restrictions on each kind of classifier.

*Component*  
A modular part of a system design that hides its implementation behind a set of external interfaces. Within a system, components satisfying the same interfaces may be substituted freely.

*Interface*  
A declaration of a coherent set of public features and obligations, a contract between providers and consumers of services.

*Object*  
An object is a discrete unit out from which the modeler understands and constructs a system. An object is an instance of a class, that is an individual with identity whose structure and behavior are described by the class. An object holds information and has a well-defined behavior and has relationships and run-time communications with other objects.

*Node*  
A UML node is a run-time physical object that represents a computational resource, which generally has at least a memory and often processing capability.

### 1.9 Study Logic

This study consists in the following activities:

- Identification of selection criteria for language and method selection for space system engineering
- Identification of relevant UML profiles to support space system requirements capture and design
- Survey of identified UML profiles
- Analysis of identified profiles as modeling languages and applicable methodologies:
  - Identification of the relationships among them.
  - The applicability in the space system avionics engineering domain context,
  - The allocation in the software lifecycle.
- Identification of a (simple) case study to assess the languages and methodology compliance to system engineering requirements
- Final assessment of methods/languages vs system engineering modeling requirements.

The logical flow of the above described activities and interdependencies are shown in Figure 1 using a UML Activity Diagram.
Figure 1  Activity diagram with study logic
2 Space System Engineering Requirements

2.1 Introduction

The objective of the study is to survey and categorize the current available UML2 profiles for system engineering in the context of space projects.

This chapter will describe which are the needs for a system modeling language for spacecrafts applications, requirements and criteria to be utilized for language selection.

They are mainly derived from ESTEC inputs and ECCS standard reference documents [A2] and [A3].

In order to support the definition/ establishment of such criteria the following basic questions/ issues have to be asked:

- What is a system in our application field?
- What are the space system project processes and relevant phase we are interesting in?
- What are the features of a space system that have to be identified?
- Which are design characteristics to be considered?

The above issues are addressed in the following sections.

2.2 Space System Engineering

A space system is constituted by a set of interacting elements which concur to the provision of a given service or product.

A given product or service can be for example, depending on the scientific application field, a set of images of a region of the earth, the provision of television channels, images of some stars, and so on.

Usually, a space system is composed by two main elements:

- The space segment (e.g. one or more satellites). This normally includes all the ground infrastructures needed to support the development of the spacecraft and the launch vehicle to deliver it into space),

- The ground segment. The latter is usually separated in two functions, one being the communication between the space and the ground (ground stations), and the other the ground infrastructure needed to control the spacecraft, to process and distribute the data. Sometimes this overall concept of ground segment is split in two parts: one is dedicated to space control, the second one is dedicated to elaboration and processing of the acquired data and distribution to the users (i.e. the data segment)
However, this architecture is a strong function of the mission product or service.

The main European Space Standard, which deals with the life cycle of a system and describes its relevant engineering processes, is the ECSS-E-10 [A.2] [A.3]. This is the reference theoretical framework in our analysis.

**Main system language characteristics:**

1. The language shall support the identification of architectural elements such as systems, subsystems, and components.
2. The language shall the identification relationships among elements.
3. The language shall the identification the functionalities realized by the elements.

### 2.3 Space System Processes and relevant Phases

The interesting phases and processes are those relevant to the definition of the System Requirements Document (SRD), on the basis of the Mission Requirements Document (MRD), in the phase A and B of a project. This means they start from the beginning of a phase A, including a SRR, ending with a system level PDR.

This process involves therefore the establishment and formalization of the system functionalities needed to implement and provide the product services required by the MRD and the establishment of a system level architecture. This architecture needs to reflect functional (subsystem or SW elements) as well as physical (HW) breakdown.

Therefore it involves an iterative top-down process, where, once the functional partitioning and identification of the main elements of the systems is accomplished; the same applies iteratively to the lower level identified elements, down to each component of the functional breakdown.

Trade-offs on system architectural levels are inherent tasks of this process, with the scope to study the feasibility of the project and to assess the capability to implement the MRD requirements. Therefore there is an intrinsic interaction between the functional requirements identification and the allocation to identified system elements.

Feedback from the system engineering process to the mission requirements, in terms of optimization, non-conformances, proposed changes are then an important activity during this phase.

The System Engineering process is not only relevant to the function definitions but also to their verification and validation (the first part and the last part of a traditional V model). Therefore the support for the verification of the system would be an added value. The reference approach should be that applied in software engineering to develop and define test cases starting form UML models.

The following figure summarize the ECSS-E10 Processes
Figure 2   ECSS E10 Processes

*Derived system language characteristics:*

R 4. The language shall support the transition from mission requirements to system level requirements (Phase A and Phase B) and the tracing among mission requirements, system requirements and system architectural elements (step wise refinement).

R 5. The language shall support the preliminary partitioning between hardware, software and manual operations and preliminary identification of subsystems.

R 6. The language shall support different scenarios and comparisons (trade offs) of several architectural solutions.

R 7. The language shall support the inherent hierarchical approach, where a system is constituted by lower level systems (step wise refinement of the system architecture), by representing the tree decomposition view of it, as well as the interrelations of lower level parts communicating with each other and with the outside environment.

R 8. The language shall support the interaction between the functional identification and the physical allocation

R 9. The language shall support for the requirement consistency and completeness to be verified
R 10. The language shall support the verification of the model and the validation part of the life cycle, maybe helping the definition of the tests.

2.4 Space System Features

Starting from the current practice, the end product of the system definition is an SRD. The proposed template in the ECSS series can be seen. Another important theoretical reference is the template included in the Mil-STD-490, used in the development of system by NASA. The logic of the system requirement development is embedded into the Table Of Contents (TOC) of the SRD, where, as an example, the functionalities of the system have to be defined separately, as well as the external interfaces, the development constraints, any safety and quality requirements.

Moreover as a first step, the functionalities of a system and their interrelationships from a functional standpoint have to be analyzed. This results in the creation of a functional system model, very similar to what in the software engineering is called a logical model.

Derived system language characteristics:

R 11. The language shall have the capability to represent/model functionalities independently from the physical allocation

R 12. The language shall have the capability to represent data flows/interactions between functionalities

R 13. The language shall have the capability to represent the system internal/external interfaces

R 14. The language shall have the capability to partition interfaces

R 15. The language shall support the structuring of requirements to represent the different SRD contents.

R 16. Tool support for the language shall provide automatic generation of the structured SRD contents.

2.5 Design Properties

Furthermore, defining a system is not simply a creative activity. Many properties are established at the beginning and further during the project execution, they can include:

- Schedule and budget constraints,
- Identification of imposed external interfaces (like for example, in spacecraft development the design constraints imposed by the selected launch vehicle)
- The technical and sometime management decision to reuse existing hardware (from the utilization of a pre-defined platform or standard components, to the reuse of existing facilities of a ground segment – e.g. ground stations)
• System performance in terms of parameters to be addressed, simulations, response statistical analysis, or other kind of analysis

• Domain specific properties to be satisfied by the architecture (e.g. avionics specific properties, tolerance, risk, cost, statistics).

In addition the resulting model should be integrated with the discipline (domain) specific methods and languages, in order to ensure feedbacks are considered. Domain specific methods are for example those supported by

• Statemate for discrete events modeling and simulation

• Simulink, or similar tools,

• Lustre/Scade for continuous processes

What actually integration means shall be further discussed, may be hooked is a better concept.

Derived system language characteristics:

R 17. The language shall support modeling of different kind of properties (such as textual annotations, criteria, any numeric values, formulas, etc.)

R 18. The language shall support reuse of modeling components

R 19. The language shall support the use of standardized building blocks and architectural elements (such as standard functions in a space segment)

R 20. The language shall enable “integration” with discipline specific methods and languages.

2.6 Other Language Features

Last but not least, we do not have to forget that “a formal language is a way to communicate ideas” and therefore, in our case shall be target to an audience of engineers, but at the same time it shall be easy to learn and to read.

Also in this exercise we need to distinguish about a language and a method, therefore it shall be also investigated, at which extent the various languages, UML profiles etc are proposing a language or a method. In any case, it is important to see in how far existing languages/methods are supported by tools, since otherwise their relevance is only limited.

Derived system language characteristics:

R 21. The language shall not require a specialist and a very expensive training on

R 22. The language shall have a simple syntax (i.e., not many symbols)

R 23. The extent at which the proposed language is strictly linked and correlated to a method, or a given process shall be evaluated
R 24. Investments in terms of learning and tool support shall be evaluated.
3 UML and Relevant Profiles

The Unified Modeling Language (UML) [R.1] is a standard object oriented modeling language from the Object Management Group (OMG).

UML is a general-purpose language that can be tailored to specific domains through a set of extension mechanisms. These mechanisms allow the customizations and extension of the UML syntax and semantic while maintaining interoperability across tools.

In mid-2001 OMG members started working on a major upgrade to UML, version 2, that currently is still on going. UML2 is a large specification, being worked in four parts (Superstructure, Infrastructure, Object Constraint Language (OCL), and Diagram Interchange).

The Superstructure specification is the most relevant part of UML2. Its adoption started in 2004, with version 2.0 and a revised submission, version 2.1, has been adopted by OMG in May 2006. UML 2.1, that is applicable the context of this study[A.4], is a revision of UML 2.0, which focuses more on tuning and bug fixing than adding new features, with no technical changes with respect to UML 2.0, which was also used for the HRT-UML study [R.11].

According to ESTEC inputs and preliminary investigations, the following profiles are identified as relevant UML profiles to support space system requirements capture and design:

- ADDL
- SysML
- MARTE
- SoC
- SystemC
- OMEGA
- TUT
4 Survey of Profiles

4.1 AADL

4.1.1 What is

The SAE Architecture Analysis and Design Language Standard (from now on called AADL) [R.2] is a language for describing the software and hardware architecture of performance-critical, distributed, real-time, embedded systems.

Performance-critical systems are systems whose operation strongly depends on meeting non-functional system requirements such as reliability, availability, timing, responsiveness, Throughput, safety, and security. The language is used to describe the structure of such systems as an assembly of software components mapped onto an execution platform. The language can describe functional interfaces to components (such as data inputs and outputs) and performance-critical aspects of components such as timing. The language can describe how components interact, such as how data inputs and outputs are connected or how application software components are allocated to execution platform components. The language can also describe the dynamic behavior of the run time architecture by supporting the modeling concept of operational modes and mode transitions. The language is designed to be extensible to accommodate analyses of the runtime architectures that the core language does not completely support. Extensions can take the form of new properties and analysis specific notations that can be associated with components.

On November 10, 2004, SAE published AADL as SAE standard AS5506. A second version of the AADL standard is supposed to be adopted by the end of the year 2006. It will integrate more advanced notions than version 1.0 which is the subject of this document.

4.1.2 Synopsis

An architecture model described in AADL is composed of both software and hardware components (hardware components are called execution platform components in the AADL standard). An architecture specification in AADL describes how components are combined in subsystems, how they interact and how they are distributed on top of hardware platforms.

AADL also describes the properties of the constituent components of a model as well as communication endpoints between these components. Extension facilities are provided in the language to accommodate the specific requirements of real-world projects.

AADL is capable of doing the following:

- Using the AADL we can describe system architecture (the containment hierarchy and connection topology of components that make up the system; with the root of this hierarchy tree representing the component that needs to be instantiated to create a working system from the blue print)

- Analyze the architecture before actually implementing it

- Describe performance critical aspects of a system
With a suitable companion runtime and translation system from AADL to a programming language, the AADL design can be transformed into a complete realtime executive + application skeleton. The application skeleton can be fleshed out with source code in a programming language to create a robust working system.

Having established clearly what AADL is useful for, we must also describe what it is not useful for, in fact following are the things for which AADL cannot be used:

- Detailed component design or implementation
- Providing detailed system behaviour information (no FSM or other descriptions of behaviour is provided)

Components are the basic building blocks of AADL models. Components can represent both software and hardware entities in the model. In total there are 10 different types of components, these are:

**Software Components**

- Data
- Subprograms and subprogram calls
- Threads
- Thread groups
- Processes

**Execution Platform Components**

- Processors
- Memories
- Buses
- Devices

**Hybrid Components**

- Systems

AADL proposes a compositional approach to system architecture, where components can contain subcomponents, each of them interacting with others to form a functioning system.

Components in AADL are specified in two parts; the component type and the component implementation.
The *type* represents the interface of the component as seen by the environment and is the only way the environment can interact with the component. *Types* contain *features* (communication endpoints) and *properties* (named attributes). *Types* may also be extended.

The *implementation* defines the internals of the component; which subcomponents it contains, how they are connected together etc. For every component *type* there can be zero or more component *implementations*. Component *implementations* may also form an extension hierarchy.

*Properties* are named attributes of *component types, component implementations, subcomponents, features, flows, connections, modes and subprogram calls*. The property name declares a name for the given property, along with its type, it also specifies which AADL elements that property can be given for.

AADL properties add semantic information about an AADL system and can be used for analysis purposes. They can be associated to component types, component implementations, subcomponents, features, connections, flows, modes and subprogram calls. When a property is associated to an AADL element, it applies to all its instances with a system instance. But AADL also supports the specification of instance specific values of any element in the containment hierarchy of a system instance.

Property types and property names are declared in property sets. They can be user-defined or predefined. There are two predeclared property sets that are applicable to all AADL specifications. User-defined properties offer one of the two extension mechanisms of the AADL language.

*Annex libraries* and *annex subclauses* offer the second extension mechanism of the AADL language where annex libraries are reusable declarations expressed in a sublanguage that are usually declared in packages and annex subclauses contain declarations expressed in a sublanguage that can be added to component types and component implementations through annexes. They can reference declarations contained in annex libraries.

*Features* are interface facilities defined in the type of a component. It is the features that allow a component to interact with the outside world. There are four categories of features:

1. Ports
2. Parameters
3. Subprograms
4. Subcomponent access

*Port* features represent a communication interface for the exchange of data and events between components. Ports are classified into data, event and event data ports, furthermore ports can have directional semantics such as in, out or in out.

The *subprogram* feature represents a call interface for a service that is accessible to other components. *Server subprogram* features represent subprograms that execute in their own thread and can be called remotely. Data subprogram features represent subprograms through which the
data component is manipulated. Call sequences specify calls to subprogram classifiers, data subprogram features, and server subprogram features.

Parameter features represent data values that can be passed into and out of subprograms. Parameters are typed with a data classifier reference.

Subcomponent access represents communication via shared access to data and bus components. A data or bus component declared inside a component implementation is specified to be accessible to components outside using the provides access feature declaration. A component may indicate that it requires access to a data or bus subcomponent declared outside using the requires access semantic in its features clause.

4.1.3 Tools
AADL Tools:

- OSATE by SEI
- Ocarina by ENST
- STOOD by TNI-Europe
- TOPCASED

An AADL OMG MOF meta-model has been recently defined on Eclipse, enabling an XML interchange format for tools supporting AADL.

The Topcased (Toolkit in Open source for Critical Application and system Development) project, supported by private and public founding and involving a large number of industries, research organizations, is currently developing Eclipse open source support for AADL and UML graphical modeling.

A UML profile is going to be defined by SAE.

4.2 SysML

4.2.1 Introduction
UML2 has taken a significant step in an attempt to add more systems engineering diagrammatic in an attempt to bridge the gap between systems and software engineering. However, as UML 2 itself is not the final answer to bridging the gap from systems to software, there are initiatives such as the SysML (Systems for UML) that are emerging to bring them closer together.

According to the International Council on Systems Engineering (INCOSE), “Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem.”
An RFP (ad/03-03-41) was published in March 2003 by OMG soliciting proposals for customizing UML for use in Systems Engineering. This RFP was drafted by the OMG Systems Engineering Domain Special Interest Group, which was jointly charted by the OMG and INCOSE in 2001.

The RFP is very detailed – here’s the introduction:

“This Request for Proposal solicits submissions that specify a customization of UML™ for Systems Engineering (SE). The customization of UML for systems engineering is intended to support modeling of a broad range of systems, which may include hardware, software, data, personnel, procedures, and facilities.

The customization of UML for SE should support the analysis, specification, design, and verification of complex systems by:

- Capturing the systems information in a precise and efficient manner that enables it to be integrated and reused in a wider context
- Analyzing and evaluating the system being specified, to identify and resolve system requirements and design issues, and to support trade-offs
- Communicating systems information correctly and consistently among various stakeholders and participants”

Specific requirements of the RFP are:

- To support modeling of system-level requirements and design, and is not intended to focus on models for detailed implementation. In the context of the OMG MDA, it should be viewed as a platform-independent modeling language.

- To provide a general-purpose systems modeling language that allow to be extended to support domain specific languages related to the application domain of the systems that are being developed or particular analysis to be supported on these systems.

- To provide support the following system modeling elements:
  - **Structural elements**, such as *system hierarchies* to model the hierarchical decomposition of a system into lower level logical or physical components *environment* elements, *system interconnections, ports, connections* between ports, *deployment* of components to nodes
  - **Behavior**, intended both as functional and state-based behavior,
  - **Properties**, that is elements having type and value, but also may include measurement units, probability distributions (e.g. mean and variance), sources, references, parametric models to support analysis models and the global time property.
  - **Requirements** to be associated to model elements of the system to express any kind of requirement, such as operational, functional, interface, performance or specialized
safety, reliability, etc. and including properties (criticality, weighting, etc.),
requirements hierarchy and traceability

- **Verifications**, that is elements to model the process used to verify the system, the
  methods used, test cases and procedures, and verification results

It as to be noted that, since this RFP was published while the UML2 specification was still evolving, many of the requirements are satisfied by UML2 without modification, in particular requirements related to the modeling of structure and behavior.

The Final Adopted SysML Specification [R.2] was developed by a broad based team including tool vendors, leading industry users, government agencies and professional organizations over a period of 3 years.

This paragraph will look at the background of SysML and summarize the modifications to UML2 and the new diagrams that are relevant to this study.

### 4.2.2 SysML Design Principles

SysML is designed to provide simple but powerful constructs for modeling a wide range of systems engineering problems. It is particularly effective in specifying requirements, structure, behavior, and allocations, and constraints on system properties to support engineering analysis. The language is intended to support multiple processes and methods, but each methodology may impose additional constraints on how a construct or diagram kind may be used.

SysML reuses a subset of UML 2.1 and provides additional extensions to satisfy the requirements of the language. SysML adds new diagrams to UML and modify others. The Venn diagram in Figure 3 shows the relationship between UML and SysML.
4.2.3 SysML Diagrams

The SysML diagram taxonomy is provided in Figure 3.
OMG SysML includes diagrams that can be used to specify system requirements, behavior, structure and parametric relationships. These are known as the four pillars of OMG SysML.

Block definition diagrams and internal block diagrams represent the system structure. A block definition diagram describes the system hierarchy and system/component classifications. The internal block diagram describes the internal structure of a system in terms of its parts, ports, and connectors. The package diagram is used to organize the model.

The behavior diagrams include the use case diagram, activity diagram, sequence diagram and state machine diagram. A use-case diagram provides a high-level description of the system functionality. The activity diagram represents the flow of data and control between activities. A sequence diagram represents the interaction between collaborating parts of a system. The state machine diagram describes the state transitions and actions that a system or its parts performs in response to events.

The requirement diagram captures requirements hierarchies and the derivation, satisfaction, verification and refinement relationships. The relationships provide the capability to relate requirements to one another and to relate requirements to system design models and test cases. The requirement diagram provides a bridge between typical requirements management tools and the system models.

The parametric diagram represents constraints on system parameter values such as performance, reliability and mass properties to support engineering analysis.

OMG SysML includes an allocation relationship to represent various types of allocation including allocation of functions to components, logical to physical components and software to hardware.

This list of diagrams is not exclusive. If a modeler feels, for example, that a UML communication diagram is necessary to communicate a particular concept, or requirement, this will be allowed.

Each OMG SysML diagram has a frame with a contents area, a heading and a diagram description as shown in Figure 4. The frame is a rectangle that is required for OMG SysML diagrams (Note: the frame is optional in UML). The frame can designate a model element that is the default namespace for the model elements enclosed in the frame. A qualified name for the model element within the frame must be provided if it is not contained within the default namespace associated with the frame. The top level “Model” name is the highest-level namespace for model elements. The frame may include border elements associated with the designated model element like ports for blocks, entry/exit points on state machines, gates on interactions, parameters for activities and constraint parameters for constraint blocks. The border of the diagram area provided by a tool may sometimes define the frame.

4.2.4 Structure

The major structural extension in OMG SysML is the «block» which extends the UML Structured Class. It is a general purpose hierarchical structuring mechanism that abstracts away much of the software-specific detail implicit in UML structured classes. Blocks can represent any level of the system hierarchy including the top-level system, a subsystem, or logical or physical component of a system or environment. An OMG SysML block describes a system as a collection of parts and
connections between them that enable communication and other forms of interaction. Ports provide access to the internal structure of a block for use when the object is used within the context of a larger structure. OMG SysML provides standard ports which support client-server communication (e.g., required and provided interfaces) and FlowPorts that define flows in or out of a block. Ports are discussed in more detail below.

Two diagrams are used to describe block relationships. The Block Definition Diagram (bdd), similar to a traditional class diagram, is used to describe relationships that exist between blocks. The Internal Block Diagram (ibd) is used to describe block internals. An example of a block definition diagram is shown in Figure 4.
An example of internal block definition diagram is shown in the following Figure 5.

Figure 6  Internal Block Diagram (ibd) Example

### 4.2.4.1 Standard Ports

Standard Ports are the same as ports in UML 2.0 and used to specify service oriented (request-reply) peer-to-peer interaction which is typical for software component architectures. Standard ports are typed by required/provided interfaces detailing the set of provided/required services. A provided interface specifies a set of operations that a block must provide and a required interface specifies a set of operations that it requires to be provided by another block.

### 4.2.4.2 Flow Ports

FlowPorts are interaction points through which data, material or energy “can” enter or leave the owning block. A FlowPort specifies the input and output items that may flow between a block and its environment. The specification of what can flow is achieved by typing the flowPort with a specification of things that flow. This can include typing an atomic flow port with a single item that flows in our out, or typing a non-atomic flow port with a “flowSpecification” which lists multiple items that flow. An atomic flowPort can be typed by a Block, ValueType, DataType or Signal. A block representing an automatic transmission in a car could have an atomic flow port that specifies “Torque” as an input and another atomic flow port that specifies “Torque” as an output. A more complex flow port could specify a set of signals and/or properties that flow in and out of the flow port. In general, flow ports are intended to be used for synchronous, broadcast or send and forget interactions. FlowPorts extend UML2.0 ports. Atomic flowPorts have an arrow inside them indicating the direction of the port with respect to the owning Block. Non-atomic flowPorts have two open arrow heads facing away from each other (i.e. <>). The fill color of the square is white and the line and text colors are black, unless the flowPort is conjugated in which case the fill color
of the square is black and the text is in white. A conjugated flowPort is the port at the opposite end. In other words, the flow specification may define a set of items flowing in and out. At the other end, the directions for these will be reversed.

4.2.5 Cross-Cutting Constructs

These apply to both structure and behavior. Cross-cutting constructs support concerns that cut across the different views and may be addressed by all or disparate parts of the model. These constructs take the form of:

- Allocations
- Requirements
- Para metrics.

Allocations define a basic allocation relationship that can be used to allocate a set of model elements to another, such as allocating behavior to structure or allocating logical to physical components.

4.2.6 Requirements

One of the two principal extensions to OMG SysML is support for requirements. Requirements are included in the Cross-cutting Constructs section which is discussed later in the paper. The «requirement» stereotype extends class to specify the textual “shall” statement and capture the requirement id#. The requirement diagram is used to integrate the system models with text based requirements that are typically captured in requirements management tools. The UML containment relationship is used to decompose a requirement into its constituent requirements. A requirement is related to other key modeling artifacts via a set of stereotyped dependencies. The «deriveReqt» and «satisfy» dependencies describe the derivation of requirements from other requirements and the satisfaction of requirements by design, respectively. The «verify» dependency shows the link from a test case to the requirement or requirements it verifies. In addition, the UML «refine» dependency is used to indicate that an OMG SysML model element is a refinement of a textual requirement, and «a copy» relationship is used to show reuse of a requirement within a different requirement hierarchy. The «rationale» concept can be used to annotate any model element to identify supporting rationale including analysis and trade studies for a derived requirement, a design or some other decision.

Only the most basic attributes of a text based requirement are included in the OMG SysML specification. More specialized requirement types can be designated using specialization of the «requirement» stereotype. Typical examples are operational, functional, interface, control, performance, physical and storage requirements. These stereotypes may restrict the types of model elements that can satisfy or refine the requirement. For example, perhaps a «performanceRequirement» can only be satisfied by a set of Constraints in a parametric diagram along with an associated tolerance and/or probability distribution, or a «functionalRequirement» might be satisfied by an activity or operation of a block. A (non-normative) set of such specialized stereotypes can be found in Appendix C of the OMG SysML specification.
The requirements model can be shown in graphical, tree structure or tabular format. An example of a requirement tree structure is shown in the following Figure 6.

**Figure 7  Requirement Tree Structure**

Additionally, requirements and their relationships can be shown on other diagrams as in Figure 7 and 8.
Figure 8  Deriving Requirements
The requirements model is not meant to replace external requirements management tools, but is meant to be used in conjunction with them to increase traceability within UML models. It could also be used for modeling the requirements and system for smaller projects. Additionally, tools that provide the functionality to import, export, and synchronize requirements and their relationships between the OMG SysML model and an external requirements management tool will allow developers to perform requirements traceability in the tool while taking advantage of the features provided by specialist requirements management tools.

4.2.7 Allocations

OMG SysML includes an allocation relationship to allocate one model element to another. Allocation is the term used by systems engineers to denote the organized cross-association (mapping) of elements within the various structures or hierarchies of a user model. Often this is the allocation of function to form, such as the deployment of software on a hardware platform, or a use case to an organization or system entity. From a systems engineering perspective, this is applicable...
to abstract system specifications rather than a particular constrained method of system or software design. Allocations can be used early in the design as a precursor to more detailed rigorous specifications and implementations. The allocation relationship can provide an effective means for navigating the model by establishing cross relationships and ensuring the various parts of the model are properly integrated. The OMG SysML specification includes some specific subclasses of allocation for allocating behavior, structure and flows, but these are given more as examples rather than an exhaustive list. A typical example is the allocation of activities to blocks (e.g., functions to components). Finally, allocation can be shown in tabular form, as well. This, of course, is only possible if the model has been built on a database and is not simply a series of unrelated diagrams.

4.2.8 Parametric Diagrams

Parametric diagrams are used to describe constraints on system properties to support engineering analysis. In order to support this type of modeling a ConstraintBlock has been introduced into OMG SysML. A ConstraintBlock defines a set of parameters and one or more constraints on the parameters. By default, these parameters are non-directional and so have no notion of causality. These ConstraintBlocks are used in a parametric diagram to constrain system properties. ConstraintBlocks may be used to express mathematical equations such as ‘\( F = m \cdot a \)’ and ‘\( a = \frac{\Delta v}{\Delta t} \)’, or statistical values and utility functions such as might be used in trade studies. Based on the reusable concept of a block new ConstraintBlocks can be built by reusing more primitive ConstraintBlocks such as basic mathematical operators.

OMG SysML also defines a model of value types that can have units and dimensions and probability distributions. The value types are used to type properties of blocks.

The Parametric Diagram is a specialized variant of an internal block diagram that restricts diagram elements to represent constraint blocks, their parameters and the block properties that they bind to. Both parameters and properties may be represented as small “pin-like” boxes to help make the diagrams more scaleable.

Examples of Parametric Diagrams are shown in Figure 9 and 10.
Figure 10  Defining properties using Parametric Diagrams
4.2.9 Behavior

The OMG SysML behavioral diagrams include the activity diagram, sequence diagram, state machine diagram and use case diagram. State machines are used to specify state-based behavior in terms of system states and their transitions. Sequence diagrams describe message based behavior of the system, subsystem, activity, use case or other owning construct. Use cases describe behavior in terms of the high level functionality and uses of a system that are further specified in the other behavioral diagrams referred to above.

These diagrams remain virtually unchanged from UML 2. Activities, which have been significantly extended from UML 2.0 activities, represent the basic unit of behavior that is used in activity, sequence and state machine diagrams. The activity diagram is used to describe the flow of control and flow of inputs and outputs among actions.

An example of SysML Activity Diagram is shown in Figure 11.
4.2.10 Continuous Systems

Additionally, OMG SysML provides extensions that might be very loosely grouped under the term “continuous” but are generally applicable to any sort of distributed flow of information and physical items through a system. These are:

- **Restrictions on the rate at which entities flow along edges in an activity, or in and out of parameters of a behavior.** This includes both discrete and continuous flows, either of material, energy or information. Discrete and continuous flows are unified under rate of flow, as is traditionally done in mathematical models of continuous change where the discrete increment of time approaches zero.

- **Extension of object nodes**, including pins, with the option for newly arriving values to replace values that are already in the object nodes. OMG SysML also extends object nodes with the option to discard values if they do not immediately flow downstream. These two extensions are useful for ensuring that the most recent information is available to actions by indicating when old values should not be kept in object nodes, and for preventing fast or
continuously flowing values from collecting in an object node, as well as modeling transient values, such as electrical signals.

4.2.11 Probabilities

OMG SysML introduces probability into activities as follows:

- Extension of edges with probabilities for the likelihood that a value leaving the decision node or object node will traverse an edge.
- Extension of output parameter sets with probabilities for the likelihood that values will be output on a parameter set.

4.2.12 Tools


The Topcased (Toolkit in Open source for Critical Application and system Development) project, supported by private and public founding and involving a large number of industries, research organizations, is currently developing Eclipse open source support for SysML and UML graphical modeling.

4.3 MARTE

4.3.1 Introduction

Marte [R.4] is an initial submission of a UML™ profile that adds capabilities for modeling Real Time and Embedded Systems (RTES), and for analyzing schedulability and performance properties of UML specifications.

This new profile is intended to replace the existing UML Profile for Schedulability, Performance and Time.

The Marte profile, should provide support for specification, design, and verification stages by:

- Providing a common way of modeling both hardware and software aspects of a RTES in order to improve communication between developers.
- Enabling interoperability between development tools used for specification, design, verification, code generation, etc.
- Fostering the construction of models that may be used to make quantitative and partitioning predictions regarding Hardware and Software characteristics.

Developed by Industry+Tool vendors+Academia: Thales, Alcatel, Lockheed Martin, Telelogic, INRIA, CEA…
4.3.2 The Approach
The profile is structured around two concerns, one to model the features of real-time and embedded systems and the other to annotate application models so as to support analysis of system properties.

4.3.3 Modelling RTE Systems
Application modelling is based on interacting component blocks for structural aspects. As for behaviour, data-intensive pipe-lined computations are generally represented with block-diagrams amenable to activity charts, while control-flow parts and communication protocols use hierarchical finite-state machines. This functionality is complemented with timing aspects, based on appropriate time/cycle descriptions (see time model section below).

Execution platform modelling comprises the description of both dedicated hardware and (middleware) software layers and interconnects composing the platform. It can be described at the same level of abstraction as the application, and contains also timing information along with structural and behavioural aspects. Explicit detailed modelling can be needed in as far as the appropriate match between application and architecture is to be studied (hierarchical cache structure or Instruction Set Simulators for instance).

The allocation model describes the association matching applicative functions onto execution platform resources. It is sometimes mandatory to provide timing information on this allocation link itself, rather that on its constituents, for reasons of modular abstraction (for instance one may indicate that a complex filter function can be realized at a given cost on a given specific processor, without going back to individual statements and instructions).

*Note: allocation is related to the similar notion in SysML.*

The MARTE profile defines precise semantics for time and resource modelling. These precise semantics allows to automatically transforming models to lower abstraction level models such as UML for SoC for hardware/software simulation or into C++ for implementation purpose.

Implementation: open-source project using Eclipse, but will need tools for e.g. performance analysis, scheduling analysis, etc. Not foreseen.

4.4 SoC
The term System-on-Chip (SoC) denotes the integration of the components of a computing or a communication system into a single chip. The components may be of digital, analog, or mixed-signal types. SoC system design is very similar to the well-known *component-based design* in software engineering. The main motivation for SoC is compactness and design reuse.

The UML profile for SoC [R.7] supports modeling and specification of SoC designs. To augment the semantics for UML-based SoC design, stereotypes for each UML metaclass are defined by introducing a special notation and constraint for each SoC element. For this purpose *SoC structure diagrams* are introduced. SoC designers create both class diagrams and SoC structure diagrams for developing an *Executable* and *Realizable* system level model.
SoC architectures are a hierarchical composition of blocks connected together by means of channels that carry data.

The SoC structure diagram is used to describe the structure of SoC. The following elements are used:

1. Module and Module Part
2. Port (Single or Multiple)
3. Protocol Interface
4. Channel Part
5. Connector
6. Protocol
7. Process
8. Clock Port
9. Reset Port
10. Clock Channel
11. Reset Channel
12. Data Type

Process behaviour is not in the scope of the profile (no action language). No model of time.

Concepts are very close to SystemC so that it enables easy code generation of the system skeleton.

4.5 System C

The UML profile for SystemC language [R.5], [R.6], based on UML 2.0, defines a language that enables to specify, analyze, design construct, visualize and document the software and hardware artifacts in SoC design flow. It does so by leveraging the joint capabilities of UML2 and SystemC to provide a modeling framework for systems in which high-level functional models can be refined down to implementation in a single platform specific language.

SystemC is a well-known object oriented system level design framework based on the C++ language, that has been incorporated into well-proven and well-understood SoC (system-on-Chip) design flows, exceeding traditional hardware description languages (such as VHDL and Verilog).
The profile captures both structure and behavior semantics of SystemC and enables SystemC code generation. In the future will allow verification of properties through model checking, thanks to a mapping of the UML profile to the formal ASM (Abstract State Machines) language.

Developed in the context of a joint project among University of Milan, University of Catania and STMicroelectronics, the profile is not proposed yet for standardization. No standard format is available. Currently tool support is provided through a customization of the Enterprise Architect tool, whose availability is unknown.

4.6 OMEGA

The aim of the OMEGA project [R.8] was the definition of a development methodology in UML for embedded and real-time systems based on formal verification techniques.

The approach was first to select a suitable subset of UML, adapted and extended where needed with a special emphasis on time related aspects.

The chosen subset includes: Class Diagrams and State Machines; Object Constraint Language (OCL); Component descriptions including provided and required interfaces; Use Case Diagrams, Live Sequence charts (an extension of UML's sequence diagrams)

UML 1.4, with anticipation of UML 2.0, in particular, Architecture diagrams with ports and connectors; Deployment Diagrams

Time extensions: based on the adopted profile for Schedulability, Time and Performance,v2. Includes special data types for time and duration, a predefined package defining timer and clock. A rich set of predefined events (which include a time stamp) and duration expressions; the possibility to define time constraints on predefined or user defined events and durations using OCL like syntax. Duration constraints based on event observations can be defined using Omega observer state machines

The second step was to propose a development methodology, based on the UML modeling and specification capabilities and the verification methods and tools developed in the project.

Starting point: simple iterative, incremental development process.

Activities for particular System Under Development, typically real-time embedded system:

1. Requirements specification and analysis
2. System analysis, definition of architecture
3. Iterate following steps, for increasing part of the System:
   i. Analysis and design of parts
   ii. Refinement until it is close to concrete implementation
iii. Implementation of (next) version on concrete platform (ultimate goal: generate the implementation from the model)

<table>
<thead>
<tr>
<th>Workflow</th>
<th>Omega / UML concepts</th>
<th>Omega Tool support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Use Case Diagrams</td>
<td>Editing and simulation of LSC</td>
</tr>
<tr>
<td></td>
<td>Live Sequence Diagrams (LSC)</td>
<td>Consistency checks for LSC and OCL</td>
</tr>
<tr>
<td></td>
<td>OCL</td>
<td>Deduction of properties from given properties</td>
</tr>
<tr>
<td></td>
<td>State Machines / Observers</td>
<td></td>
</tr>
<tr>
<td>Architecture</td>
<td>Components, Connectors, Ports, Required &amp; Provided Interfaces, OCL constraints, (Protocol) State Machines</td>
<td>Correctness wrt requirements Compositional verification Timing Analysis</td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td>Class Diagrams, OCL, State Machines</td>
<td>Correctness wrt component specs Synthesis of state diagrams Correctness of refinement steps, within and between iterations</td>
</tr>
<tr>
<td>Platform dependent model</td>
<td>Deployment Information (resources, scheduling policies, execution time constraints</td>
<td>Timing verification and synthesis of constraints</td>
</tr>
</tbody>
</table>

OMEGA was completed in February 2004. At that time UML2 was not available, so it is difficult to relate it to UML2.

### 4.7 TUT

The TUT-Profile is a UML2.0 profile defined by the Tampere University of Technology for large embedded systems design. The profile defines a set of UML stereotypes that extend the UML2.0 metamodel with standard extension mechanisms. Thus the TUT-Profile is compatible with commercial UML2.0 tools. TUT-Profile is based on IP-block granularity and libraries for platform components. The new feature is to keep all essential design information explicitly at UML2.0 level without any auxiliary languages, models or descriptions.

The TUT-Profile divides system modeling into the design of application, architecture and mapping models. The models also contain non-functional constraints for specification, design automation, analysis and implementation. The models also include back-annotated performance values from real implementation or simulation.
The KOSKI design framework, that transforms UML2.0 specifications to Multi-Processor SoC implementations, supports the profile. It includes automated steps for program code generation, heterogeneous IP-block architecture exploration, application distribution among processors with RTOS integration and MP-SoC platform assembly. From the implementation real-time performance and memory measurements are back-annotation to the UML2.0 models for constraints matching.

The design flow is entirely governed by UML2.0 models with TUT-Profile. UML2.0 models and library components are used as such without separate manual programming or conversions.

Figure 13 show the TUT profile hierarchy.

![TUT Profile Hierarchy](image)

Figure 13 TUT-Profile hierarchy for UML 2.0 design.

The following table shows a summary of TUT stereotypes.

<table>
<thead>
<tr>
<th>Stereotype name</th>
<th>Metaclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>Class</td>
<td>Top-level application class</td>
</tr>
<tr>
<td>ApplicationComponent</td>
<td>Class</td>
<td>Functional application component (active class, has behavior)</td>
</tr>
<tr>
<td>ApplicationProcess</td>
<td>Structural feature</td>
<td>Instance of a functional application component (part in architecture diagram)</td>
</tr>
<tr>
<td>ProcessGroup</td>
<td>Structural feature</td>
<td>Group of application processes</td>
</tr>
<tr>
<td>ProcessGrouping</td>
<td>Dependency</td>
<td>Grouping dependency between application processes and process groups</td>
</tr>
<tr>
<td>Platform</td>
<td>Class</td>
<td>Top-level platform class</td>
</tr>
<tr>
<td>PlatformComponent</td>
<td>Class</td>
<td>Platform component class defining features of a platform component</td>
</tr>
<tr>
<td>PlatformComponentInstance</td>
<td>Structural feature</td>
<td>Instantiated platform component</td>
</tr>
<tr>
<td>CommunicationWrapper</td>
<td>Dependency</td>
<td>Defines wrapper parameters of a communication agent</td>
</tr>
<tr>
<td>CommunicationSegment</td>
<td>Structural feature</td>
<td>Interconnection structure of communicating agents</td>
</tr>
<tr>
<td>PlatformMapping</td>
<td>Dependency</td>
<td>Dependency between a process group and a platform component instance</td>
</tr>
</tbody>
</table>
The profile is not proposed for standardization. No standard format is available. The availability of the support tool is unknown.
5 Analysis of Profiles

5.1 Introduction

This section is intended to analyze the different profiles, describe their relationships and allocation in the space software life cycle.

5.2 Relationships among profiles

SysML is a general-purpose OMG standard language that extends UML2 for system engineering. It may cover system engineering, from high–level requirements to architecture and verification, spanning over different domains. SysML may act as a unifying base language to describe space systems and be easily related to UML for software modeling.

AADL is a language to describe system and software architectures focused on the avionics domain, specialized to describe performance-critical, distributed, real-time embedded systems. AADL currently lacks of support for modeling functional and dynamic behavior as well as support for modeling variability in system families. However, an AADL UML profile may be defined as a specialized subset of the SysML profile.

The SoC OMG adopted profile is a language to support specific SoC component design and enabling SystemC code generation. The SystemC and TUT profiles address the same area, but they are not standards, so that they should be considered mainly as reference modeling languages for SoC development.

OMEGA is focused in the area of software architectures for embedded and real-time systems, with emphasis on formal verification techniques. Being not a standard and based on UML1 and the old profile for Schedulability, Performance and Time, OMEGA will be in the future superseded by MARTE so that, again, OMEGA should be considered mainly as reference modeling languages in his area.

The MARTE profile, being targeted to embedded and real-time software systems, declares the intention to integrate with both SysML at the upper level for the system architecture and SoC at the lower level for component design and code generation. Together with SysML, it competes with the AADL language in the avionics domain. Anyway MARTE currently is defined by an initial specification submission only, whose contents are still not mature enough.

5.3 Allocation of Profiles to the software life cycle

Figure 14 shows the envisioned UML2 unified modeling approach.
Figure 14 The UML2 unified modeling approach
# 6 SysML Compliance to System Engineering Requirements

<table>
<thead>
<tr>
<th>ID</th>
<th>Requirement</th>
<th>Compliance</th>
</tr>
</thead>
<tbody>
<tr>
<td>R 1.</td>
<td>The language shall support the identification of architectural elements such as systems, subsystems, and components.</td>
<td>Blocks, blocks description diagrams, internal block diagrams.</td>
</tr>
<tr>
<td>R 2.</td>
<td>The language shall support the identification relationships among elements.</td>
<td>Dependencies, compositions, connectors inside internal block diagrams.</td>
</tr>
<tr>
<td>R 3.</td>
<td>The language shall the identification the functionalities realized by the elements.</td>
<td>Block naming, association of activity diagrams describing the functionalities, associated to requirement blocks.</td>
</tr>
<tr>
<td>R 4.</td>
<td>The language shall support the transition from mission requirements to system level requirements (Phase A and Phase B) and the tracing among mission requirements, system requirements and system architectural elements (step wise refinement).</td>
<td>Representation of two separate hierarchies of requirements, representation of the system decomposition tree, derived dependency to trace among the different requirement sets and the satisfy dependency to trace from architectural elements to requirements.</td>
</tr>
<tr>
<td>R 5.</td>
<td>The language shall support the preliminary partitioning between hardware, software and manual operations and preliminary identification of subsystems.</td>
<td>Block description diagrams with the containment hierarchy.</td>
</tr>
<tr>
<td>R 6.</td>
<td>The language shall support different scenarios and comparisons (trade offs) of several architectural solutions.</td>
<td>Different containment hierarchies and allocations of the requirements to the system elements by means of the derived dependency.</td>
</tr>
<tr>
<td>R 7.</td>
<td>The language shall support the inherent hierarchical approach, where a system a constituted by lower level systems (step wise refinement of the system architecture), by representing the tree decomposition view of it, as well as the interrelations of lower level parts communicating with each other and with the outside environment.</td>
<td>Block description diagrams containing the containment hierarchy that defines the product tree, internal block diagrams describing the collaboration among system parts.</td>
</tr>
<tr>
<td>R 8.</td>
<td>The language shall support the interaction between the functional identification and the physical allocation.</td>
<td>Allocation of activity diagrams to blocks</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Compliance</td>
</tr>
<tr>
<td>----</td>
<td>--------------</td>
<td>------------</td>
</tr>
<tr>
<td>R 9.</td>
<td>The language shall support for the requirement consistency and completeness to be verified.</td>
<td>System requirements consistency and completeness can be verified through the &lt;&lt;deriveReq&gt;&gt; dependencies, each mission requirement has to derive at lest by a system requirement element, and vice versa, each system requirement has to be derived by at lest a mission requirement.</td>
</tr>
<tr>
<td>R 10.</td>
<td>The language shall support the verification of the model and the validation part of the life cycle, maybe helping the definition of the tests.</td>
<td>Model consistency and completeness can be verified through the &lt;&lt;satisfy&gt;&gt; dependencies, each requirement has to be satisfied at lest by a system element, and vice versa, each system element has to satisfy at lest a requirement. Test case elements can be associated to the block elements, to support validation.</td>
</tr>
<tr>
<td>R 11.</td>
<td>The language shall have the capability to represent/model functionalities independently from their allocation to system elements.</td>
<td>Functionalities are represented by means of activity diagrams separately from system elements.</td>
</tr>
<tr>
<td>R 12.</td>
<td>The language shall have the capability to represent data flows/interactions between functionalities.</td>
<td>Activity diagrams.</td>
</tr>
<tr>
<td>R 13.</td>
<td>The language shall have the capability to represent the system internal/external interfaces</td>
<td>Blocks, blocks description diagrams, internal block diagrams with external interfaces and delegation.</td>
</tr>
<tr>
<td>R 14.</td>
<td>The language shall have the capability to partition interfaces</td>
<td>Multiple interfaces may be associated to blocks.</td>
</tr>
<tr>
<td>R 15.</td>
<td>The language shall support the structuring of requirements to represent the different SRD contents.</td>
<td>Structuring of requirements is supported by means of requirement blocks, the structuring can be easily organized and further modified.</td>
</tr>
<tr>
<td>R 16.</td>
<td>Tool support for the language shall provide automatic generation of the structured SRD contents.</td>
<td>This is a tool capability..</td>
</tr>
<tr>
<td>R 17.</td>
<td>The language shall support modeling of different kind of properties (such as textual annotations, criteria, any numeric values, formulas, etc.).</td>
<td>Properties and parametric diagrams can be associated to block elements.</td>
</tr>
<tr>
<td>R 18.</td>
<td>The language shall support modeling of reused components</td>
<td>The language supports mechanisms to for domain analysis and design and reuse of components. When used with UML, providing object oriented mechanisms to model variability, it can provide complete support for system families development in the different system phases.</td>
</tr>
<tr>
<td>ID</td>
<td>Requirement</td>
<td>Compliance</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>R 19.</td>
<td>The language shall support the use of standardized building blocks and architectural elements (such as standard functions in a space segment).</td>
<td>See above.</td>
</tr>
<tr>
<td>R 20.</td>
<td>The language shall enable “integration” with discipline specific methods and languages.</td>
<td>The possible integrations and the specific support provided by tools should be investigated.</td>
</tr>
<tr>
<td>R 21.</td>
<td>The language shall not require a specialist and a very expensive training on.</td>
<td>The principles of SysML are easy to understand.</td>
</tr>
<tr>
<td>R 22.</td>
<td>The language shall have a simple syntax (i.e., not many symbols)</td>
<td>SysML is a quite simple graphic language.</td>
</tr>
<tr>
<td>R 23.</td>
<td>The extent at which the proposed language is strictly linked and correlated to a method, or a given process shall be evaluated</td>
<td>A specific method needs to be defined.</td>
</tr>
<tr>
<td>R 24.</td>
<td>Investments in terms of learning and tool support shall be evaluated</td>
<td>Not addressed in the context of this study.</td>
</tr>
</tbody>
</table>
7 Using SysML for Space System and Software Engineering

7.1 Using SysML for Space System Engineering

In relation to the new ECSS-E10 standard, SysML may be used to support the following system engineering functions:

- **requirement engineering** - in this activity SysML provides specific support to describe the system and lower level functional and technical specifications, in particular by helping to organize and visualize the requirements structuring and providing navigable traceability among different level related requirements, among requirements and related implementing blocks, among requirements and related test cases and among requirements and related object types.

- **system analysis** - in this activity SysML provides specific support to consolidate requirements through trade-off and various analysis (mission, requirement, functional, physical, performance), and produce in particular the functional and physical architecture

- **system design** - in this activity SysML provides specific support to produce the functional and physical architecture, and to describe properties of the system elements

- **system configuration** - in this activity SysML provides specific support to produce the product tree of the system in terms of physical and software elements

- **system verification** - in this activity SysML provides specific support to define test cases for the system to be conformant to requirements

- **system integration** - in this activity SysML provides specific support to ensure system engineering control and data interfaces, including static and dynamic aspects for nominal and degraded modes and data medium used for data exchange.

7.2 A Model for the System Engineering Processes

The model of the system engineering is partitioned into a collection of high level models each representing the outputs of the main areas of the system engineering process. The SysML diagram in Figure 15 shows the system engineering models and the relationships among them.
7.3 Using SysML for Software Related System Engineering

In relation with the new ESCC-E40C, the SysML language may be used to support system and software co-engineering processes.

A specific system specification methodology using SysML should be defined and enforced to include definitions for the following software related models and properties:

- Computational model
- Data model
- Functional model
- Event model
- Failure model
- Logical safety, aliveness, timeliness and schedulability properties

Note that specific UML diagrams may be more appropriate for some of the above models. However the SysML language specification does not prevent the use of UML for some specific purposes.

7.4 Moving from System Engineering to Software Engineering

Software related system engineering models needs to be connected with software engineering languages that are suitable for the software architecture engineering, design, implementation and verification processes, according to ECSS E40.
According to the defined system specification methodology, as mentioned in the above paragraph, and to the language/methodology selected for software engineering, a mapping needs to be defined to move from the system to the initial software specification.

The mapping should be defined for the following system model elements:

- **requirement models related with software** – software functional and technical specifications, by maintaining the traceability to high level related requirements, related implementing blocks and test cases.
- **system design models related with software** - the software related functional and physical architecture, including specific properties and constraints
- **system verification models related with software** - the software related test cases
- **system integration and control models related with software** - the control and data interfaces, including static and dynamic aspects for nominal and degradated modes and data medium used for data exchange.
- **other software related models and properties** - as mentioned in the above paragraph.

For example a model mapping from these models to the ASSERT methodology should be defined.

Insurance that the software specification and architecture are directly produced from the system specification may be granted by a model to model automatic transformation, according to a model-based engineering approach, and verification supported through traceability links from the system to the software model elements.

An initial definition for a methodology integrating system and software engineering is shown in figure
Figure 16 System Engineering Processes and relations to Sw Engineering
8 Requirements for SysML Tool Support

In order to enable the assessment of tool support as currently available for SysML, the next step in the context of this study is the analysis of the requirements we envisage for such tools.

8.1 General Requirements

In principle SysML should be considered as a new language.

Although it reuses several UML 2.0 diagrams and their semantics and it reduces the size of the language in terms of the number of diagrams, based on the ways in which a UML language can be extended, it also introduces new diagrams and extends the capabilities of some of the existing diagrams.

But above all, what is more relevant of SysML as a system language, is that it makes it easy to interpret all diagrams from a systems engineering point of view.

So, in conclusion SysML is strictly a "new" language, even though it's not using UML diagrams in a way that its semantics is drastically changed.

From the perspective of a software tool implementing SysML, there may be several different perspectives:

- Since SysML uses most of UML2 elements and introduces new elements that are derived from the existing ones, an existing UML2 tool can be customized through a SysML profile of UML 2.0 for the domain of systems engineering.

- Existing UML2 tools can be customized through a SysML profile of UML2 by direct support for the new diagrams such as requirement and parametric diagrams, and the modified diagrams, that are activity, block and internal block diagrams. They can also allow the other kind diagrams not required by SysML, that are communication, class and component and deployment diagrams.

- New tools can be developed that are specific for SysML alone; they may still be based on a profile of UML2, or be specifically developed on the SysML proper UML subset and extensions.

All the different perspectives have advantages and disadvantages.

In principle, solutions based on profiling of existing UML2 tools may not be easy to use for system engineers, if the underlying details of the UML2 complex syntax are not hidden to the user. In this case that the easiness of interpretation of SysML diagrams would not correspond to their easy production.

For example, when drawing an internal block diagram, users need to be presented with a diagram toolbar and behavior that is specific to the IBD, rather than the underlying composite structure diagram with properties, so that IBD can be easily created, according to the SysML language constraints.
Moreover elements in a SysML model are closely related. For instance constraints in a parametric diagram may depend on other constraints, properties have mandatory links with blocks which needs, as a consequence, a specific link between IBDs and BDDs. This kind of relationships needs to be maintained and the specific SysML constraints have to be applied.

Hence, we think that a profile can't cover all the SysML mechanisms and we think that a specific metamodel, the related editor and consistency checks are more appropriate to apply the whole SysML philosophy, not the use of a UML2 editor profile feature.

So a ‘deep’ level of SysML support would enhance usability, while retaining full conformance to the standard, which anyway is based on a UML profile.

In conclusion we envisage:

- The availability of specific diagram editors that provide a tailored palette of tools to easy create SysML elements and their connections, according to the SysML syntax and semantics rules;

- Specific support for UML restrictions and extensions constraints, by enforcing in this way also the SysML semantics.

The advantages with UML2 tools customized through a SysML profile is that all the UML diagrams are still available so that different models or packages can be defined using SysML diagrams at system engineering level and UML diagrams at software level using the same tool so that traceability links among the different levels can be defined by the user.

On the other end solutions specifically developed for SysML could be very easy to use, but their integration with UML more expensive to realize.

In any case SysML tools should be based on the OMG standards, such as MOF and XMI, and possibly developed as Eclipse plug-ins. In this way they could offer the best approach in terms of feasibility of integration among SysML and UML tool support and enable model based engineering with automatic transformations from the system level to the software level design.

**Derived tool generic requirements**

TR1 Tool support for SySML shall be based on specialized editors for the SysML diagrams that are easy to use and do not require any UML knowledge.

TR2 Tool support for SySML shall be based on specialized editors for the SysML diagrams that enforce the SysML syntax and semantics.

TR3 Specialized editors shall be provided at list for the following SysML diagrams:

- Requirement diagram
- Block definition diagram
- Internal block diagram
8.2 Support for Requirement Modeling

In the SysML specification authors have explicitly declared that the SysML requirement modeling constructs are only intended to provide a bridge between traditional requirements management tools and the other SysML models.

We expect at least SysML tools to be integrated with requirements management and provide bi-directional migration of requirements, with emphasis on creation/editing of requirements and automatic generation of documentation on the requirement management tool side and structuring and tracing of requirements on the SysML tool side.

Alternatively SysML tools should provide support for requirements to be edited in requirement tables diagrams, where a tabular format is used to represent the requirements, their properties and relationships. In this case SysML tools should also provide all the facilities requirement management tools usually support, including the automatic generation of specification documents.

Derived requirements for requirement modeling

TR5 Tool support for SySML shall integrate with requirement management off-the-shelf solutions

TR6 Tool support for SySML shall import requirements created with requirement management off-the-shelf solutions

TR7 Tool support for SySML shall import/export editing of requirements with requirement management off-the-shelf solutions

TR8 Tool support for SySML shall import/export structuring of requirements with requirement management off-the-shelf solutions

TR9 Tool support for SySML shall be integrated with requirement management off-the-shelf solutions, so that automatic generation of specification documents can be enabled, with contents that are structured according to SYSML requirement modeling.

TR10 Tool support for SySML shall import requirements created with requirement management off-the-shelf solutions
TR11 As alternative to requirements TR1-TR9, tool support for SySML shall provide all the requirement management facilities directly, as from requirement management off-the-shelf solutions, including generation of specification documents as from TR8.

8.3 **Support for Block Diagrams**

TR12 Block definition diagram and internal block diagram editors shall present toolbars and behavior that is specific to BDDs and IBDs, rather than the underlying composite structure diagram with properties, so that BDDs and IBDs can be easily created, according to the SysML language constraints

TR13 Graphical editing of hierarchies of blocks elements shall be supported through BDD by showing composition relationships (as in class diagrams)

TR14 Graphical editing of hierarchies of blocks elements shall be supported through IBD showing containment relationships

TR15 Graphical editing of hierarchies of blocks elements shall be consistent among BDDs and related IBDs

TR16 Hierarchies of block elements shall be represented internally in a way that is consistent with diagram editing.
9 Conclusions

SysML today seems to offer the best support to system engineering than available from other so-called Architectural Design Languages (ADL), such as AADL:

- It provides full support for all the system engineering processes, from Requirement Engineering and Analysis to Design and Verification Engineering, and “reflect” the current space practices and needs;

- In connection with UML, or with specialized UML profiles, a “common” path can be easily identified, to address system and software model-based engineering together, so that a stronger relation can be found from system level to software level processes, according to the ASSERT approach;

- In connection with UML, or with specialized UML profiles, it can address also domain engineering, so that system families can be modeled, with reference to the ASSERT approach;

- In addition, SysML is easy to use, well accepted by users and from available technical reports, defined according to the widespread used industrial OMG standards, and finally, supported by a large list of commercial and free tools.

Further envisaged activities are:

- Investigations and in-house experiments using space projects documentation and practical and on the field exercises are needed to confirm these preliminary results

- Identification of requirements for CASE support and assessment of the tools available on the market.

- Definition of a concrete and detailed method for its practical use in space system engineering, by addressing aspects that are still not well covered in this note such as:
  - detailed relationship to the system and software life cycles and to the E10 and e40 processes;
  - investigation on the definition of parametric diagrams and constraints for use in system engineering, with reference also to AADL available properties, with analysis purposes;

- Investigation on integrations of SysML (and related tool support) with other discipline specific methods and languages, such as those used for thermal, mechanical, control and other kind of software analysis and simulations.