TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Software & Systems Engineering
Prof. Dr. Dr. h.c. Manfred Broy

SPES 2020 Deliverable D1.2.B-6

SPES Metamodel

Software Plattform Embedded Systems 2020

Author:	Alexander Harhurin
	Florian Hölzl
	Thomas Kofler
Version:	1.0
Date:	December 15, 2010
Status:	Released

Technische Universität München - Fakultät für Informatik - Boltzmannstr. 3 - 85748 Garching

Contents

1 Introduction

1.1 Two dimensions of abstraction

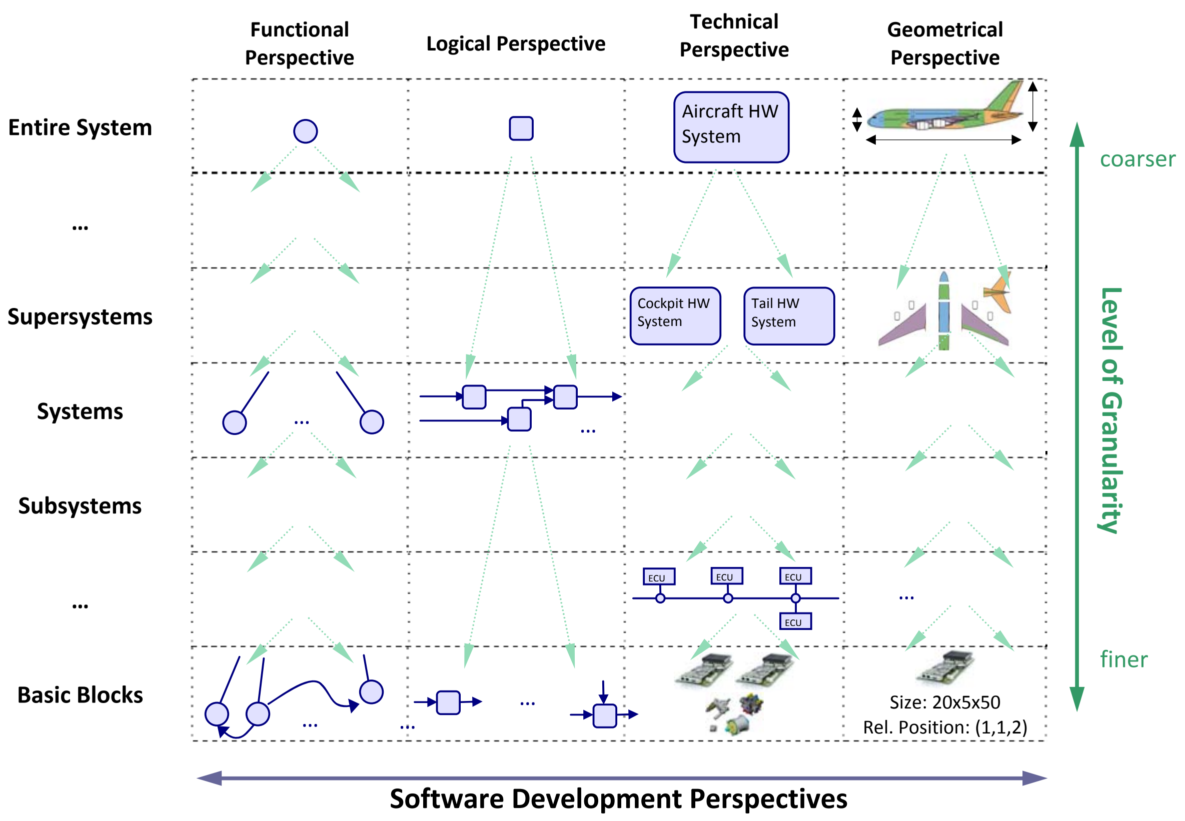

Figure 1: Two dimensions of abstraction: granularity levels and development perspectives [TRS+10]

1.2 Glossary

Levels of granularity: A system is composed of sub-systems which are at a lower granularity level and which can themselves be regarded as systems. Each sub-system can be considered as a system itself and can be further decomposed until we reach basic building blocks (cf. Figure 1).

Software development perspectives: A system can be regarded from different perspectives, each perspective representing different kinds of information about the system (cf. Figure 1).

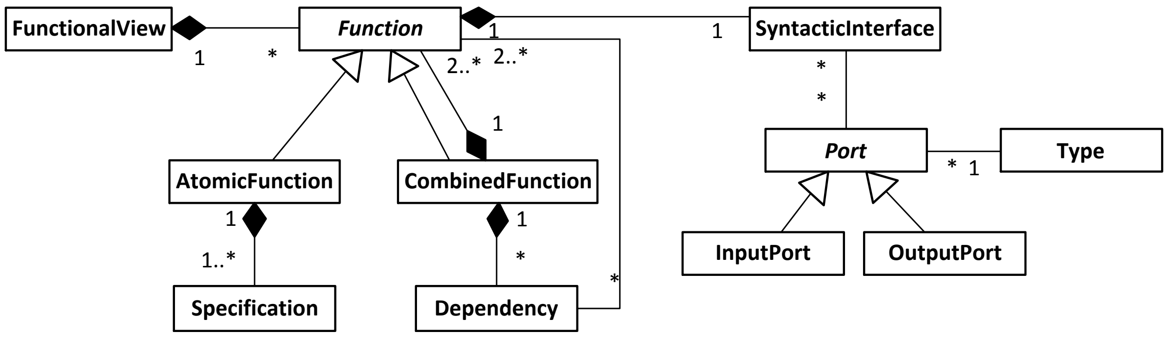

Figure 2: Functional Perspective

2 Functional Perspective

Functional View (or Perspective): The functional perspective (cf. Figure 2) describes the usage functionality that a system offers its environment/users (cf. [BFG⁺08, TRS⁺10, BH09, Har10]). The central aims of the functional perspective are: Hierarchical structuring of the functionality from the point of view of the system's users; Definition of the boundary between the system functionality and its environment: definition of the syntactical interface and abstract information flow between the system and its environment; Consolidation of the functional requirements by formally specifying the requirements on the system behavior from the black-box perspective; Understanding of the functional interrelationships and mastering of feature interaction.

Function: The functional view consists of several functions. A function is a syntactical and/or semantical projection of the overall system functionality. A function is defined as a mapping from a subset of inputs to a subset of outputs. A function has a syntactical interface and may be atomic or combined.

Atomic Function: An atomic function is given by one or more behavior specifications.

Combined Function: A combined function consists of two or more functions. Combined functions are used to build a functions hierarchy.

Specification: A specification defines the behavior of a function and may be given by an I/O automaton (cf. [BH09, Har10]), an I/O table (cf. [TH09]), . . .

Dependency: By dependency relations, we mean relations between functions in a way that the operation of one function depends on those of other functions. There are a lot of methodological significant dependency relations like enables, modifies or priority. Dependencies may be given e.g., by an I/O automaton (cf. [BH09, Har10]) or implemented by external modes (cf. [Bro07]).

Syntactic Interface: A function has a syntactic interface consisting of input and output ports.

Port: A port is an interaction point between the system and its environment.

Type: Each port has a data type, e.g., boolean or integer. Note, that more sophisticated type systems, e.g., including physical units (m, s^2, $\dots$) are thinkable.

Input Port: The system receives messages from its environment through its input ports.

Output Port: The system sends messages into its environment through its output ports.

3 Logical Perspective

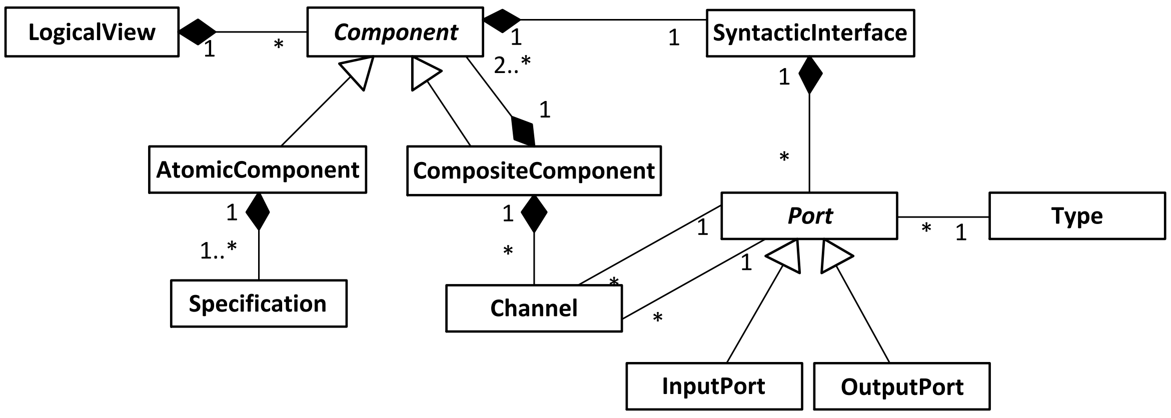

Figure 3: Logical Perspective

Logical View (or Perspective): The logical perspective (cf. Figure 3) describes the application of the system by a network of communicating and cooperating components (cf. [BS01]). The central aims of the logical perspective are:

- Hierarchical structuring of the application by means of components.
- Definition of the syntactical interfaces and information flow between the system and its environment (input and output communication) and between the different subsystems (local communication).
- Implementation of the system functionality and its interaction with the environment by means of behavior specifications.

Component: The logical view consists of a set of components. A component is a syntactical and semantical projection of the overall system with the outermost component giving the interface between the system and its environment. A component is mathematically defined as a mapping from the valuations of input stream histories to the valuations of the output stream histories (cf. [BS01]). A component has a syntactical interface and may be atomic or combined.

Atomic Component: An atomic component is given by one or more (alternative) behavior specifications describing the input/output behavior of that component.

Composite Component: A composite component consists of a network of two or more components (each being atomic or, again, a composite). Composite components are used to structure the system architecture into smaller units and connecting them with communication links (called channels). The behavior of the composite component is defined by the parallel composition of the behaviors of the sub-components (cf. [BS01]).

Specification: A specification defines the behavior of a component and may be given by an I/O automaton or an I/O table or even more specific descriptions like filters or PID controllers.

Syntactic Interface: A component has a syntactic interface consisting of input and output ports.

Port: A port is an interaction point between the system and its environment. Note, that in contrast to the functional perspective any port has at most one source of information, i.e., atomic output ports receive the data to be sent from the atomic behavior, while composite output ports are connected to at most one atomic output port via a channel. Analogously, atomic input ports receive their data from a composite input port or an atomic output port, respectively.

Type: Each port has a data type, e.g., boolean or integer. Note, that more sophisticated type systems, e.g., including physical units (m, s^2, ...) are thinkable.

4 Technical Perspective

Technical View (or Perspective): The technical perspective describes the hardware or virtual machine platform the system is to be executed on. Figure 4 shows the generic hardware meta-model (in black) and two levels of domain-specific extensions thereof (in red). Due to the plethora of available hardware and VM solutions the SPES technical meta-model is intended to be extended for a given application domain or even for a given kind of application (e.g. engine control, in-flight entertainment, virtual power plant network).

The central aims of the technical perspective are:

- Hierarchical structuring of the processing and communication units to be used as execution platform.
- Specific description of specialized peripheral components (like sensors and actuators).
- Specific description of communication networks.

Computing Unit: A computing unit provides the application with processing power allowing algorithms and control programs to be executed. Each computing unit contains various types of ports to access its environment. Furthermore, each computing unit may have an internal structure decomposing it into smaller computing and transmission units, e.g., a set of electronic control units in a car are connected via a CAN field bus, while some of the ECUs can be decomposed into specific behavioral elements connected via a network on the chip.

Transmission Unit: A transmission unit connects multiple computing units via transceiver ports.

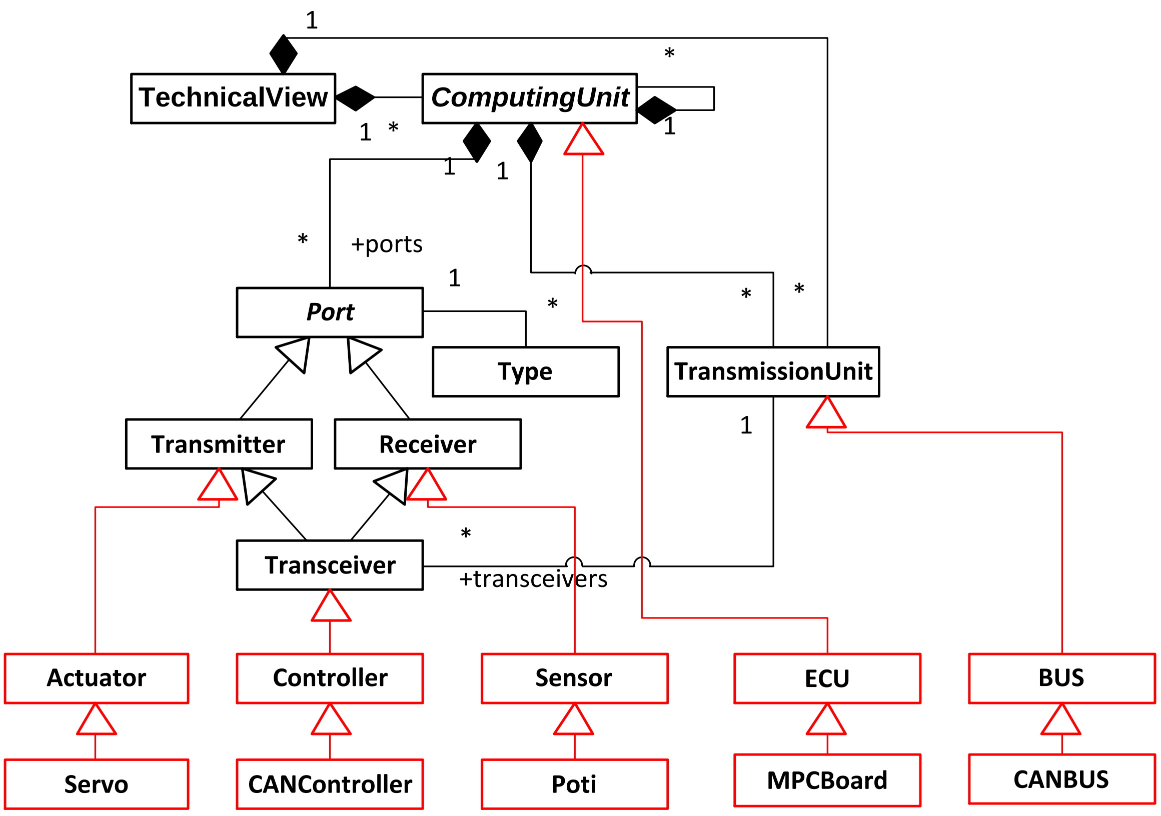

Figure 4: Technical Perspective

Port: A port is a generalization of of hardware units or virtual machine parts that connect computing units with their environment and allows data exchange.

Type: Ports are usually typed with low-level types like integer or boolean or with high-level types like bus protocols or more complex data structures.

Transmitter: A computing unit can provide information to its environment via transmitter ports. With respect to the logical perspective the output ports of logical components are mapped to transmitter ports, e.g., some integer value is transmitted to servo-motor resulting in a position change of the driver window.

Receiver: A computing unit can gather information from its environment via receiver ports. With respect to the logical perspective the input ports of logical components are mapped to receiver ports, e.g., the analog signal of a potentiometer is converted to a digital value and provided to the application as an integer value.

Transceiver: A transceiver allows the computing unit to send and receive information. Transceiver ports are connected to some transmission unit. They usually represent network driver software provided by the operating system or the virtual machine of the computing unit.

References

[BFG$^+$08] Manfred Broy, Martin Feilkas, Johannes Grünbauer, Alexander Gruler, Alexander Harhurin, Judith Hartmann, Birgit Penzenstadler, Bernhard Schätz, and Doris Wild. Umfassendes architekturmodell für das engineering eingebetteter software-intensiver systeme. Technical Report TUM-I0816, Technische Universität München, june 2008.

[BH09] Jewgenij Botaschanjan and Alexander Harhurin. Integrating Functional and Architectural Views of Reactive Systems. In *Proceedings of CBSE'09*, volume 5582 of *LNCS*. Springer, 2009.

[Bro07] Manfred Broy. A theory for requirements specification and architecture design of multi-functional software systems. *Mathematical Frameworks for Component Software: Models for Analysis and Synthesis*, pages 119–154, 2007.

[BS01] Manfred Broy and Ketil Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* Springer, 2001.

[Har10] Alexander Harhurin. *Von separaten Interaktionsmustern zu konsistenten Spezifikationen reaktiver Systeme.* PhD thesis, Technische Universität München, 2010.

[TH09] Judith Thyssen and Benjamin Hummel. Behavioral Specification of Reactive Systems Using Stream-Based I/O Tables. In *7th IEEE International Conference on Software Engineering and Formal Methods (SEFM)*. IEEE Computer Society, November 2009.

[TRS$^+$10] Judith Thyssen, Daniel Ratiu, Wolfgang Schwitzer, Alexander Harhurin, Martin Feilkas, and Eike Thaden. A system for seamless abstraction layers for model-based development of embedded software. In *Proceedings of Envision 2020 Workshop*, 2010.