Wrocław University of Technology

Chair of Cybernetics and Robotics

Component based approach for distributed control system

Mariusz Janiak
p. 331 C-3, 71 320 26 44

Contents

1. Component-Based Software Engineering

2. SysML

Component-Based Software Engineering

Component-Based Software Engineering (CBSE) is an approach that has arisen in the software engineering community in the last decade. It aims to shift the emphasis in system building from traditional requirement analysis, system design, and implementation to composing software systems from a mixture of reusable off-the-shelf and custom-built components[1]. It emerged from the failure of object-oriented development to support effective reuse. Single object classes are too detailed and specific. Components are more abstract than object classes and can be considered to be stand-alone service providers.

[1] D. Brugali, A. Shakhimardanov, *Component-Based Robotic Engineering (Part II)*, in Robotics & Automation Magazine, IEEE, vol.17, no.1, pp.100-112, March 2010

Component-Based Software Engineering

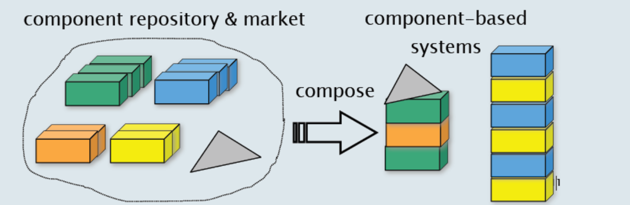
CBSE is said to be primarily concerned with three functions[2]

- Developing software from pre-produced parts
- The ability to reuse those parts in other applications
- Easily maintaining and customizing those parts to produce new functions and features

[2] G. T. Heineman, W. T. Councill, *Component-based software engineering : putting the pieces together*. Addison-Wesley, Boston, 2001.

Component-Based Software Engineering

Application development becomes the selection, adaptation and composition of components rather than implementing the application from scratch.



[3]

[3] A. I. Khan, N. -ul-Qayyum, U. A. Khan, *An Improved Model for Component Based Software Development*, in Software Engineering, Vol. 2 No. 4, 2012, pp. 138-146

Component-Based Software Engineering

CBSE essentials

- Independent components specified by their interfaces
- Component standards to facilitate component integration
- Middleware that provides support for component inter-operability
- A development process that is geared to reuse

Component-Based Software Engineering

Benefits of reuse

- Increased Reliability – components already exercised in working systems
- Reduced Process Risk – less uncertainty in development costs
- Effective Use of Specialists – reuse components instead of people
- Standards Compliance – embed standards in reusable components
- Accelerated Development – avoid custom development and speed up delivery

Component-Based Software Engineering

Apart from the benefits of reuse, CBSE is based on following software engineering design principles

- Components are independent so do not interfere with each other
- Component implementation is hidden
- Communication is through well-defined interfaces
- Component platforms are shared and reduce development costs

Component-Based Software Engineering

Component-based software design of distributed systems

- Requirements specification
- Component analysis
- System design with reuse (component selection/development, routing connections, parametrization)
- Deployment on application servers
- Instantiation and monitoring at runtime

Component-Based Software Engineering

Component definitions

- Councill and Heinmann – *A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.*
- Szyperski – *A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.*

Component-Based Software Engineering

Component as a service provider

- Components provide a service without regard to where the component is executing or its programming language
- The component is an independent, executable entity that can be made up of one or more executable object classes
- It does not have to be compiled before it is used with other components
- The services offered by a component are made available through an interface
- All component interactions take place through that interface

Component-Based Software Engineering

Component characteristics

- Standardised – component standardization means that a component that is used in a CBSE process has to conform to some standardised component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.
- Independent – a component should be independent, it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a 'requires' interface specification.

Component-Based Software Engineering

Component characteristics (cont.)

- Composable – For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.
- Deployable – To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.

Component-Based Software Engineering

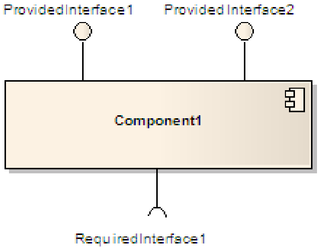
Component characteristics (cont.)

- Explicit context dependencies – specification of the deployment and run-time environments. Which tools, platforms, resources, other components are required?
- Documented – components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

Component-Based Software Engineering

Component interfaces

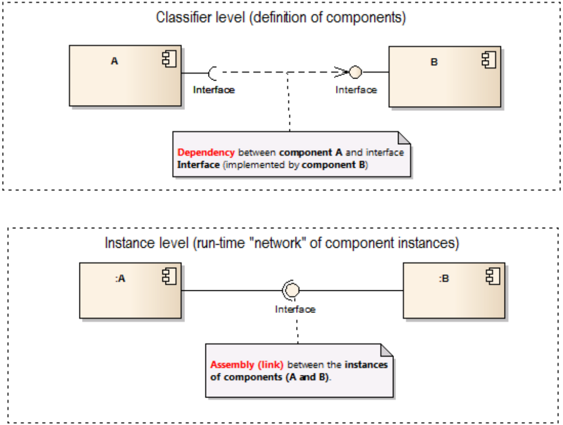
- Provides interface – defines the services that are provided by the component to other components
- Requires interface – Defines the services that specifies what services must be made available for the component to execute as specified



[4]

[4] Wikipedia

Component interfaces (cont.)


Classifier level (definition of components)
A
Interface
Interface
B
Instance level (run-time "network" of component instances)
:A
Interface
:B

Dependency between **component A** and interface **Interface** (implemented by **component B**)

Assembly (link) between the **instances of components (A and B)**.

[5]

[5] www.stackoverflow.com

Component-Based Software Engineering

Contract (part of the interface specification)

- A specification attached to an interface that mutually binds the clients and providers of the components
- Functional Aspects (API)
- Pre- and post-conditions for the operations specified by API
  - Preconditions denoting the constraints which need to be met by the client
  - Postconditions denoting the constraints which the component promises to fulfil in return
  - A component may also additionally satisfy global constraints called invariants
- Non-functional aspects (different constrains, environment requirements, etc.)

Component-Based Software Engineering

Components and objects

- Components are deployable entities
- Components do not define types
- Component implementations are opaque
- Components are language-independent
- Components are standardised

Component-Based Software Engineering

Component models

- A component model is a definition of standards for component implementation, documentation and deployment
- Examples of component models – EJB model (Enterprise Java Beans), .NET model, Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition
- Different application domains have different needs for component-based systems (different non-functional properties: performance, security, reliability, scalability, etc)

Component-Based Software Engineering

Middleware support

- Component models are the basis for middleware that provides support for executing components
- Component model implementations provide
  - Platform services that allow components written according to the model to communicate
  - Horizontal services that are application-independent services used by different components
- To use services provided by a model, components are deployed in a container. This is a set of interfaces used to access the service implementations.

Component-Based Software Engineering

Robotics Component Oriented middlewares

- OROCOS
- ORCA
- OpenRTM
- Player
- ROS (not strict follow CBSE principles)

SysML

What is SysML

- A graphical modelling language developed in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233a
- Supports the specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- **Is** a visual modeling language that provide
  - Semantics = meaning, connected to a metamodel (rules governing the creation and the structure of models)
  - Notation = representation of meaning, graphical or textual
- **Is not** a methodology or a tool (SysML is methodology and tool independent)

SysML

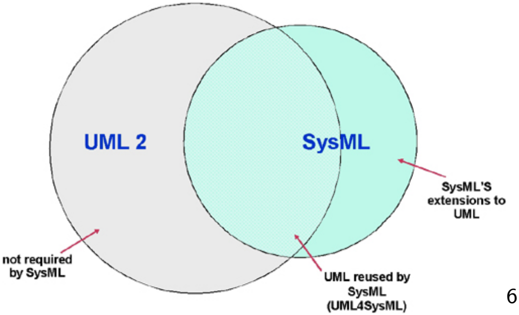
SysML vs UML

- UML is a general-purpose graphical modeling language aimed at Software Engineers
- Diagrams not used – Object diagram, Deployment diagram, Component diagram, Communication diagram, Timing diagram and Interaction overview diagram
- Diagrams from UML – Class diagram (Block Definition Diagram, Class → Block), Package diagram, Composite Structure diagram (Internal Block Diagram), State Machine Diagram, Activity Diagram, Use Case Diagram, Sequence Diagram

SysML

SysML vs UML (cont.)

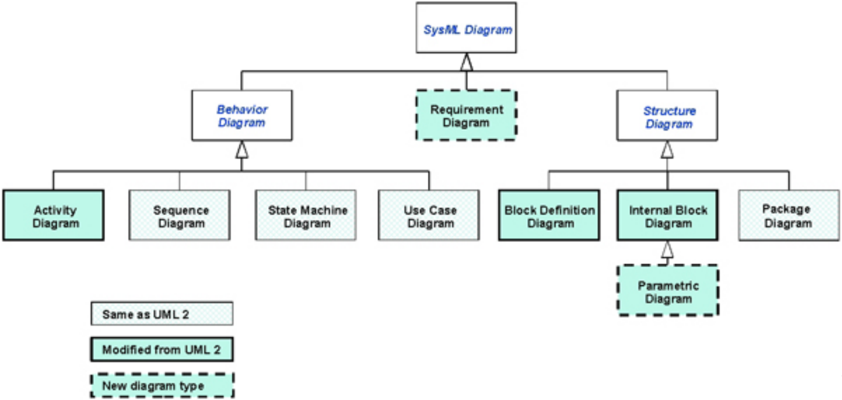
- In addition, SysML adds some new diagrams and constructs – Parametric diagram, Requirement diagram, Flow ports, Flow specifications, Item flows, Allocation



[6]

[6] http://www.omgsysml.org

SysML Diagram Types



SysML Diagram

Behavior Diagram

Requirement Diagram

Structure Diagram

Activity Diagram

Sequence Diagram

State Machine Diagram

Use Case Diagram

Block Definition Diagram

Internal Block Diagram

Package Diagram

Parametric Diagram

Same as UML 2

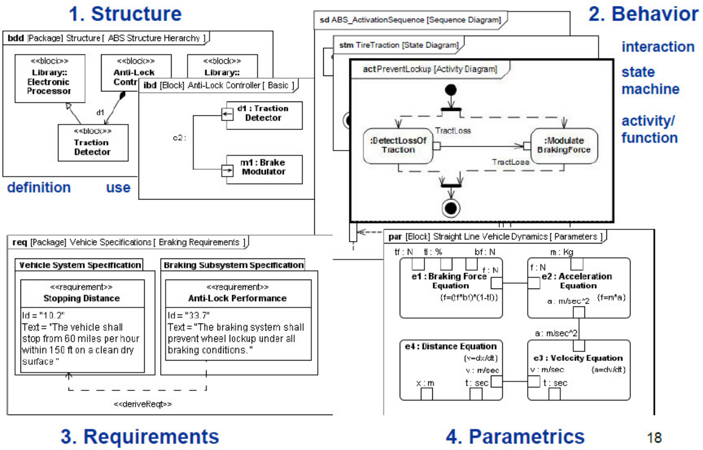
Modified from UML 2

New diagram type

[7]

[7] http://www.omgsysml.org

SysML


1. Structure
bdd [Package] Structure [ABS Structure Hierarchy]
<<block>> Library:: Electronic Processor
<<block>> Anti-Lock Contr
<<block>> Library::
d1
<<block>> Traction Detector
ibd [Block] Anti-Lock Controller [Basic]
d1 : Traction Detector
c2 :
m1 : Brake Modulator
definition
use
sd ABS_ActivationSequence [Sequence Diagram]
stm TireTraction [State Diagram]
act PreventLockup [Activity Diagram]
:DetectLossOf Traction
TractLoss
:Modulate BrakingForce
TractLoss
2. Behavior
interaction
state machine
activity/ function
req [Package] Vehicle Specifications [Braking Requirements]
Vehicle System Specification
<<requirement>> Stopping Distance
Id = "10.2"
Text = "The vehicle shall stop from 60 miles per hour within 150 ft on a clean dry surface"
Braking Subsystem Specification
<<requirement>> Anti-Lock Performance
Id = "33.7"
Text = "The braking system shall prevent wheel lockup under all braking conditions."
<<deriveReqt>>
3. Requirements
par [Block] Straight Line Vehicle Dynamics [Parameters]
tf : N
tl : %
bf : N
m : Kg
e1 : Braking Force Equation
f : N
{f=(tf*bf)*(1-tl)}
f : N
e2 : Acceleration Equation
a : m/sec^2
{f=m*a}
a : m/sec^2
e4 : Distance Equation
{v=dx/dt}
v : m/sec
x : m
t : sec
c3 : Velocity Equation
v : m/sec
{a=dv/dt}
t : sec
4. Parametrics
18

8

[8] http://www.omgsysml.org

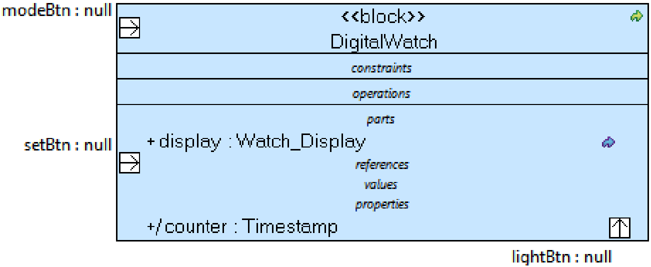
SysML

Blocks – basic structural elements

- Based on UML Class from UML Composite Structure (supports unique features e.g. flow ports, value properties)
- Provides a unifying concept to describe the structure of an element or system
- Any type of system/element – hardware, software, data, procedure, facility, person, signal, physical quantity
- Compartments are used to describe the block characteristics – Properties (parts, references, values, ports), operations, constraints allocations from/to other model elements (e.g. activities), requirements the block satisfies, user defined compartments

SysML

Blocks and Compartments



[9]

[9] http://www.omgsysml.org

SysML

Block Diagrams

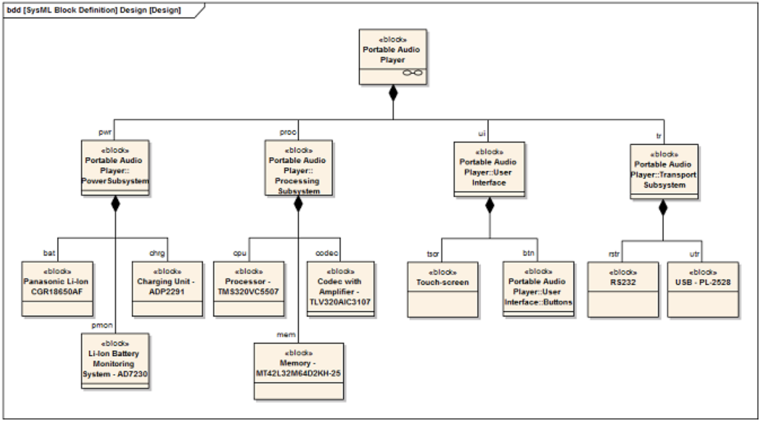
- Blocks Used to Specify Hierarchies and Interconnection
- **Block definition diagrams** describe the relationship among blocks (e.g., composition, association, specialization)
- **Internal block diagrams** describe the internal structure of blocks in terms of properties and connectors

SysML

Block Definition Diagram (BDD)

- The (Block) BDD is the same as a type definition
- Captures properties, relations, dependencies . . .
- Reused in multiple contexts
- The BDD cannot define completely the communication dependencies and the composition structure (no topology)

Block Definition Diagram (BDD)


bdd [SysML Block Definition] Design [Design]
«block» Portable Audio Player
pwr
proc
ui
tr
«block» Portable Audio Player:: PowerSubsystem
«block» Portable Audio Player:: Processing Subsystem
«block» Portable Audio Player::User Interface
«block» Portable Audio Player::Transport Subsystem
bat
chrg
cpu
codec
tscr
btn
rstr
utr
«block» Panasonic Li-Ion CGR18650AF
«block» Charging Unit - ADP2291
«block» Processor - TMS320VC5507
«block» Codec with Amplifier - TLV320AIC3107
«block» Touch-screen
«block» Portable Audio Player::User Interface::Buttons
«block» RS232
«block» USB - PL-2528
pmon
mem
«block» Li-Ion Battery Monitoring System - AD7230
«block» Memory - MT42L32M64D2KH-25

[10]

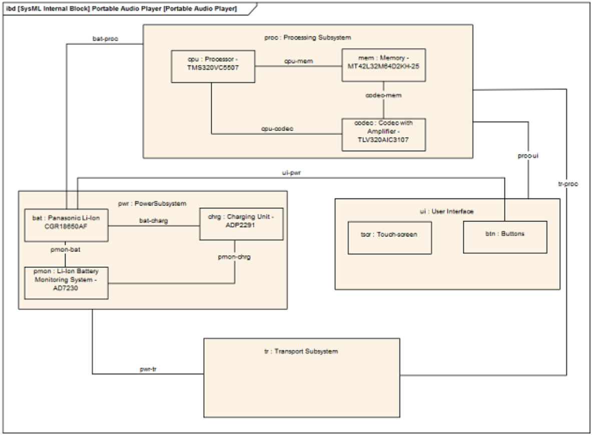
[10] http://www.sparxsystems.com

SysML

Internal Block Diagram (IBD)

- Defines the use of Blocks in a composition
- Part is the usage of a block in the context of a composing block (also known as a role)
- The internal structure becomes explicit
- The communication and signalling topology becomes explicit

Internal Block Diagram (IBD)


ibd [SysML Internal Block] Portable Audio Player [Portable Audio Player]
proc : Processing Subsystem
cpu : Processor - TMS320VC5507
cpu-mem
mem : Memory - MT42L32M64D2KH-25
codec-mem
cpu-codec
codec : Codec with Amplifier - TLV320AIC3107
bat-proc
proc-ui
ui-pwr
tr-proc
pwr : PowerSubsystem
bat : Panasonic Li-Ion CGR18650AF
bat-charg
chrg : Charging Unit - ADP2291
pmon-bat
pmon-chrg
pmon : Li-Ion Battery Monitoring System - AD7230
ui : User Interface
tscr : Touch-screen
btn : Buttons
tr : Transport Subsystem
pwr-tr
[11]

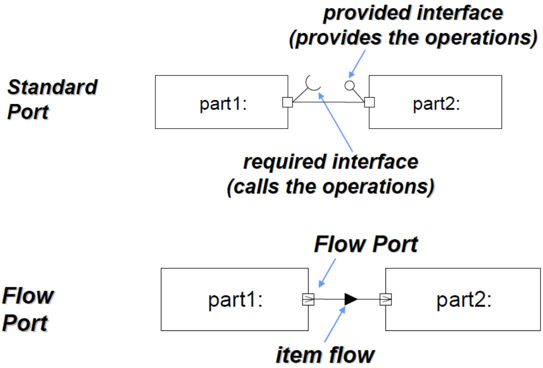
[11] http://www.sparxsystems.com

SysML

SysML Ports

- Specify interaction points on blocks and parts
- Integrate behavior with structure
- Syntax: portName:TypeName
- Kinds of ports
  - Standard (UML) Port – operation oriented for SW components; specifies a set of required or provided operations and/or signals; typed by a UML interface
  - Flow Port – used for signals and physical flows; specifies what can flow in or out of block/part; typed by: a block, value type, or flow specification
- Standard Port and Flow Port support different interface concepts
- Flow Port deprecated since SysML v1.3 (proxy and full port)

Port notation



Standard Port

provided interface (provides the operations)

part1:

part2:

required interface (calls the operations)

Flow Port

Flow Port

part1:

part2:

item flow

[12]

[12] http://www.omgsysml.org

The End

Thank you for your kind attention.