Politechnika Wrocławska

Katedra Cybernetyki i Robotyki

Podejście zorientowane na komponenty

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

Zawartość wykładu

Inżynieria oprogramowania – komponenty

Inżynieria oprogramowania zorientowana na komponenty (*Component-Based Software Engineering* CBSE) jest podejściem wypracowanym w ostatnim dziesięcioleciu w środowisku inżynierów oprogramowania. Ma ona na celu przesunięcie nacisku w procesie budowania systemu, który tradycyjnie składa się z analizy wymagań, projektowania systemu i wdrożenia, na komponowanie systemu z mieszanki gotowych, dostępnych na rynku komponentów wielokrotnego użytku, uzupełnionych dedykowanymi komponentami wytworzonymi na potrzeby opracowywanej aplikacji.[1]

[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

Inżynieria oprogramowania – komponenty

Jednym z powodów rozpoczęcia prac nad podejściem zorientowanym na komponenty było fiasko metodologi programowania obiektowego w szczególności w zakresie ponownego wykorzystania wcześniej opracowanych implementacji klas. Obiekty klas są zbyt szczegółowe i konkretne. Komponenty są bardziej abstrakcyjne niż obiekty klas i można je traktować jako niezależnych dostawców usług.

Inżynieria oprogramowania – komponenty

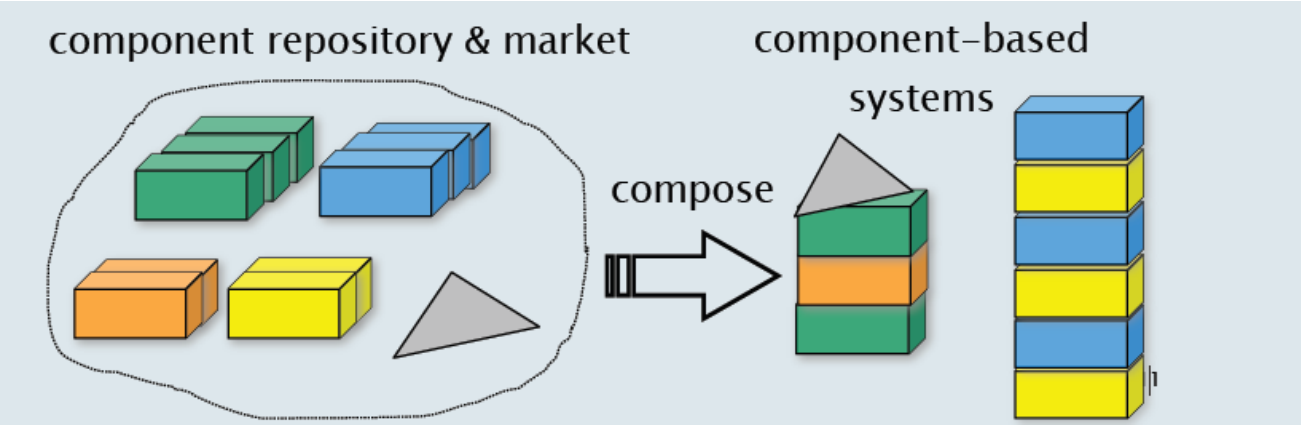
Uważa się, że CBSE stawia sobie trzy zasadnicze cele[2]

- Rozwijanie oprogramowania z prefabrykowanych elementów
- Ponowne wykorzystanie opracowanych wcześniej elementów w innych aplikacjach
- Łatwe utrzymanie i dostosowanie opracowanych elementów do nowych zadań

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

Inżynieria oprogramowania – komponenty

Rozwój aplikacji polega na selekcji, adaptacji i składaniu elementów, a nie na implementacji aplikacji od podstaw.



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

Inżynieria oprogramowania – komponenty

CBSE podstawy

- Niezależne komponenty zdefiniowane przez ich interfejsy
- Standardy umożliwiające integrację komponentów
- *Middleware* umożliwiające działanie i współpracę komponentów
- Proces projektowania zorientowany na wykorzystanie wcześniej opracowanych komponentów

Inżynieria oprogramowania – komponenty

Zalety użytkowania gotowych elementów

- zwiększona niezawodność – komponenty zostały już sprawdzone w działających systemach,
- zmniejszone ryzyko procesowe – mniejsza niepewność w kosztach rozwoju,
- skuteczne wykorzystanie specjalistów – ponowne wykorzystanie komponentów, a nie specjalistów,
- zgodność z normami – standardy wbudowane w komponenty,
- przyspieszony rozwój – unikanie tworzenia dedykowanego kodu.

Inżynieria oprogramowania – komponenty

Oprócz korzyści związanych z ponownym wykorzystaniem, CBSE opiera się na następujących zasadach projektowania oprogramowania

- komponenty są niezależne, nie kolidują ze sobą,
- implementacja komponentu jest ukryta,
- komunikacja odbywa się za pośrednictwem dobrze zdefiniowanych interfejsów,
- środwisko dla komponentów jest wspólne, ogólnodostępne, przez co zmniejsza się koszty rozwoju.

Inżynieria oprogramowania – komponenty

Procedura projektowanie systemu rozproszonego bazująca na CBSE

- specyfikacja wymagań,
- analiza „komponentowa”,
- projektowanie systemu w oparciu o komponenty (wybór/implementacja komponentów, trasowanie połączeń, parametryzacja),
- wdrożenie na serwerach aplikacji (*deployment*),
- uruchomienie i monitorowanie podczas pracy.

Inżynieria oprogramowania – komponenty

Definicja komponentu

- Councill and Heinmann – *Komponent jest elementem oprogramowania zgodnym z ustalonym modelem, który może być niezależnie uruchamiany i łączony z innymi komponentami bez konieczności modyfikacji zgodnie ze standardem kompozycji komponentów.*
- Szyperski – *Komponent jest jednostką składową oprogramowania zawierającą deklaratywnie określone interfejsy i jasno sprecyzowane zależności. Komponenty można uruchamiać niezależnie, mogą być wykorzystywane przez osoby trzecie.*

Inżynieria oprogramowania – komponenty

Komponent jako dostawca usług

- dostarcza ustalone usługi niezależnie od miejsca jego uruchomienia i wykorzystanego do implementacji języka programowania,
- jest niezależną, wykonywalną jednostką, która może być jednym lub składową wielu obiektów klas,
- nie musi być kompilowany, by móc użytkować go z innymi,
- usługi są dostępne za pośrednictwem interfejsów
- wszystkie interakcje odbywają się za pośrednictwem interfejsów

Inżynieria oprogramowania – komponenty

Charakterystyka komponentów

- Standaryzacja – komponent określony przez CBSE musi być zgodny z wybranymi znormalizowanymi modelami. Ten model może definiować interfejsy, meta-dane, dokumentację, sposób kompozycji i uruchamiania.
- Niezależność – komponent powinien być niezależny, powinno być możliwe jego uruchomienie i łącznie z innymi bez konieczności używania dodatkowych specyficznych komponentów. W sytuacjach, w których komponent wymaga zewnętrznych usług, powinny być one wyraźnie określone w specyfikacji interfejsu („wymaga").

Inżynieria oprogramowania – komponenty

Charakterystyka komponentów (cd.)

- Komunikacja – dla komponentu wszystkie interakcje zewnętrzne muszą odbywać się za pośrednictwem interfejsów zdefiniowanych publicznie. Ponadto, komponent powinien dostarczać publicznej informacji o sobie, np. dostępne metody i atrybuty.
- Uruchamianie – komponent musi być samowystarczalny, musi być w stanie działać jako samodzielna jednostka w środowisku które implementuje ustalony model komponentowy. Zazwyczaj oznacza to, że komponent występuje w formie binarnej, przez co nie musi być kompilowany przed uruchomieniem.

Inżynieria oprogramowania – komponenty

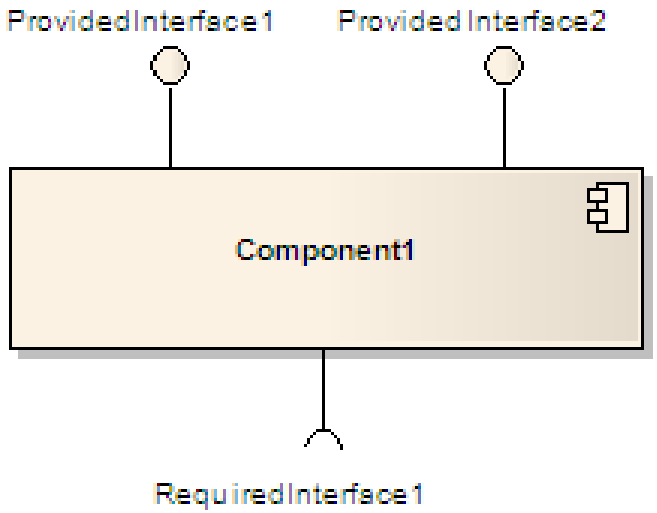
Charakterystyka komponentów (cd.)

- Predefiniowane zależności – specyfikacja środowiska (*midleware*). Jakie narzędzia, platformy, zasoby i inne składniki są wymagane?
- Dokumentacja – komponenty muszą być w pełni udokumentowane, aby potencjalni użytkownicy mogli zdecydować, czy zaspokajają ich potrzeby. Składnia, idealnie również semantyka, wszystkich interfejsów musi być zdefiniowana.

Inżynieria oprogramowania – komponenty

Interfejsy komponentu

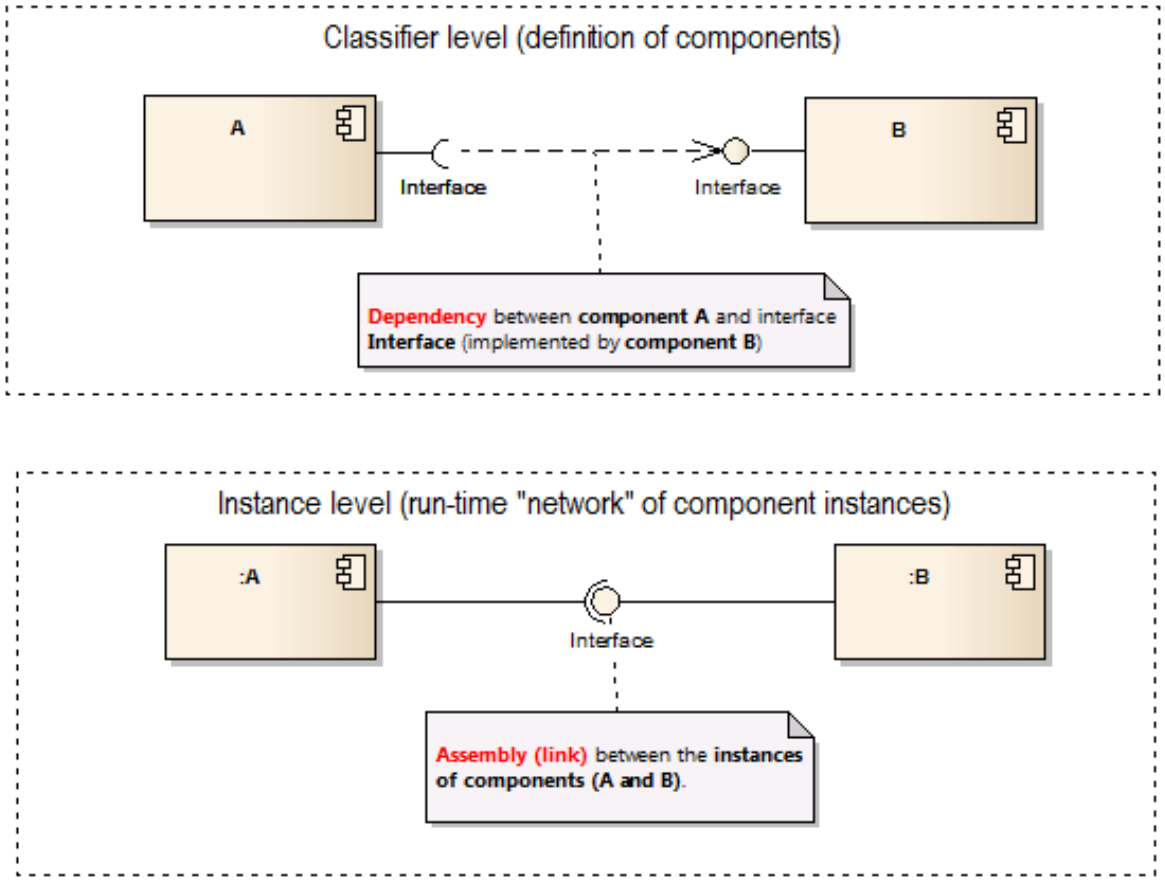
- Dostarcza – definiuje usługi dostarczane przez komponent do innych komponentów
- Wymaga – definiuje usługi, które muszą być udostępnione dla danego komponentu, by ten działał zgodnie ze specyfikacja



[4] Wikipedia

Inżynieria oprogramowania – komponenty

Interfejsy komponentu (cd.)



5

[5] www.stackoverflow.com

Inżynieria oprogramowania – komponenty

Kontrakt (część specyfikacji interfejsu)

- Specyfikacja dołączona do interfejsu wzajemnie wiążąca klientów i dostawców
- Aspekty funkcjonalne (API)
- Warunki pre/post funkcji API
  - Warunki wstępne oznaczające ograniczenia, które muszą być spełnione przez klienta
  - Warunki końcowe oznaczające ograniczenia, które komponent obiecuje spełnić w zamian
  - Komponent może również dodatkowo spełniać globalne ograniczenia zwane niezmiennikami
- Aspekty poza funkcjonalne (inne ograniczenia, środowiska, wymagania itp.)

Inżynieria oprogramowania – komponenty

Komponent a obiekt

- Komponent można uruchomić
- Komponenty nie definiują typów
- Implementacja komponentu jest ukryta
- Komponenty są niezależny od języków programowania
- Komponenty są standaryzowane

Inżynieria oprogramowania – komponenty

Modele komponentów

- Model komponentu to definicja standardów: implementacji, dokumentacji i wdrożenia komponentów.
- przykładowe modele komponentów – EJB (Enterprise Java Beans), .NET, Corba Component Model
- Model komponentu określa sposób definiowania interfejsów i elementy, które powinny być zawarte w definicji interfejsu
- Różne potrzeby w odniesieniu do systemów zbudowanych w oparciu o komponenty (wydajność, bezpieczeństwo, niezawodność, skalowalność itp.)

Inżynieria oprogramowania – komponenty

Wsparcie dla komponentów – *middleware*

- Model komponentu stanowi bazę oprogramowania pośredniczącego umożliwiającego uruchamianie komponentów
- Implementacja wybranego modelu dostarcza
  - usługi umożliwiające wzajemną komunikację komponentów
  - Usługi horyzontalne, które są niezależnymi od aplikacji usługami używanymi przez różne komponenty
- Aby korzystać z usług świadczonych przez model, komponenty są uruchamiane w kontenerze. Jest to zestaw interfejsów używanych do uzyskiwania dostępu do implementacji usług.

Inżynieria oprogramowania – komponenty

Robotyczne oprogramowanie pośredniczące zorientowane na komponenty

- OROCOS
- ORCA
- OpenRTM
- Player
- ROS ??? (podejście agentowe)

SysML

Co to jest SysML

- Graficzny język modelowania powstały jako odpowiedź na *UML for Systems Engineering RFP* opracowany przez OMG, INCOSE i AP233a
- Wspiera specyfikację, analizę, projektowanie, weryfikację i walidację systemów obejmujących sprzęt, oprogramowanie, dane, personel, procedury i urządzenia
- **Jest** graficznym językiem modelowania specyfikującym
  - Semantykę – znaczenie w odniesieniu do meta-modelu (zasady rządzące tworzeniem i strukturą modeli)
  - Notacja – reprezentacja znaczenia, graficzna lub tekstowa
- **Nie jest** metodologią lub narzędziem (SysML jest niezależny od metodologi i narzędzi)

SysML

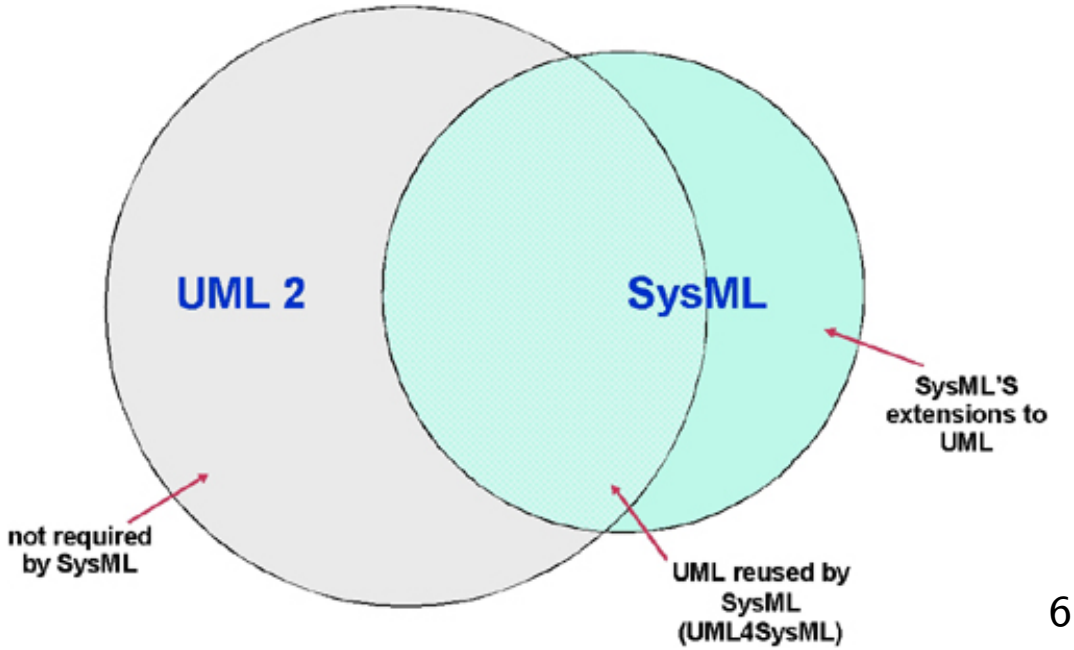
SysML vs UML

- UML jest uniwersalnym, graficznym językiem modelowania, skierowanym do inżynierów oprogramowania
- Nieużywane diagramy – Object diagram, Deployment diagram, Component diagram, Communication diagram, Timing diagram and Interaction overview diagram
- Diagramy z UML – Class diagram (Block Definition Diagram, Class $\rightarrow$ Block), Package diagram, Composite Structure diagram (Internal Block Diagram), State Machine Diagram, Activity Diagram, Use Case Diagram, Sequence Diagram

SysML

SysML vs UML (cd.)

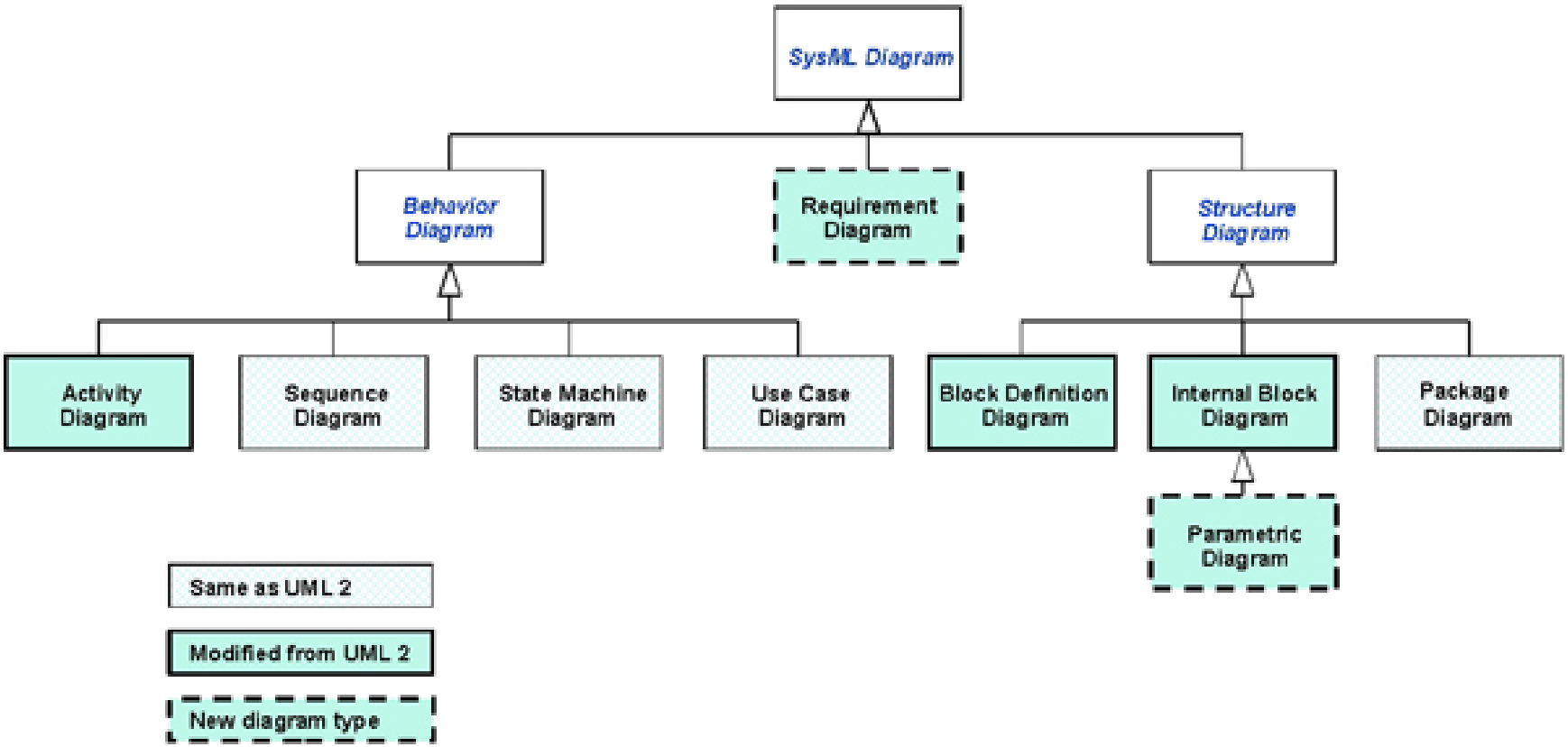
- Dodatkowo, SysML wprowadza nowe diagramy i konstrukty – Parametric diagram, Requirement diagram, Flow ports, Flow specifications, Item flows, Allocation



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

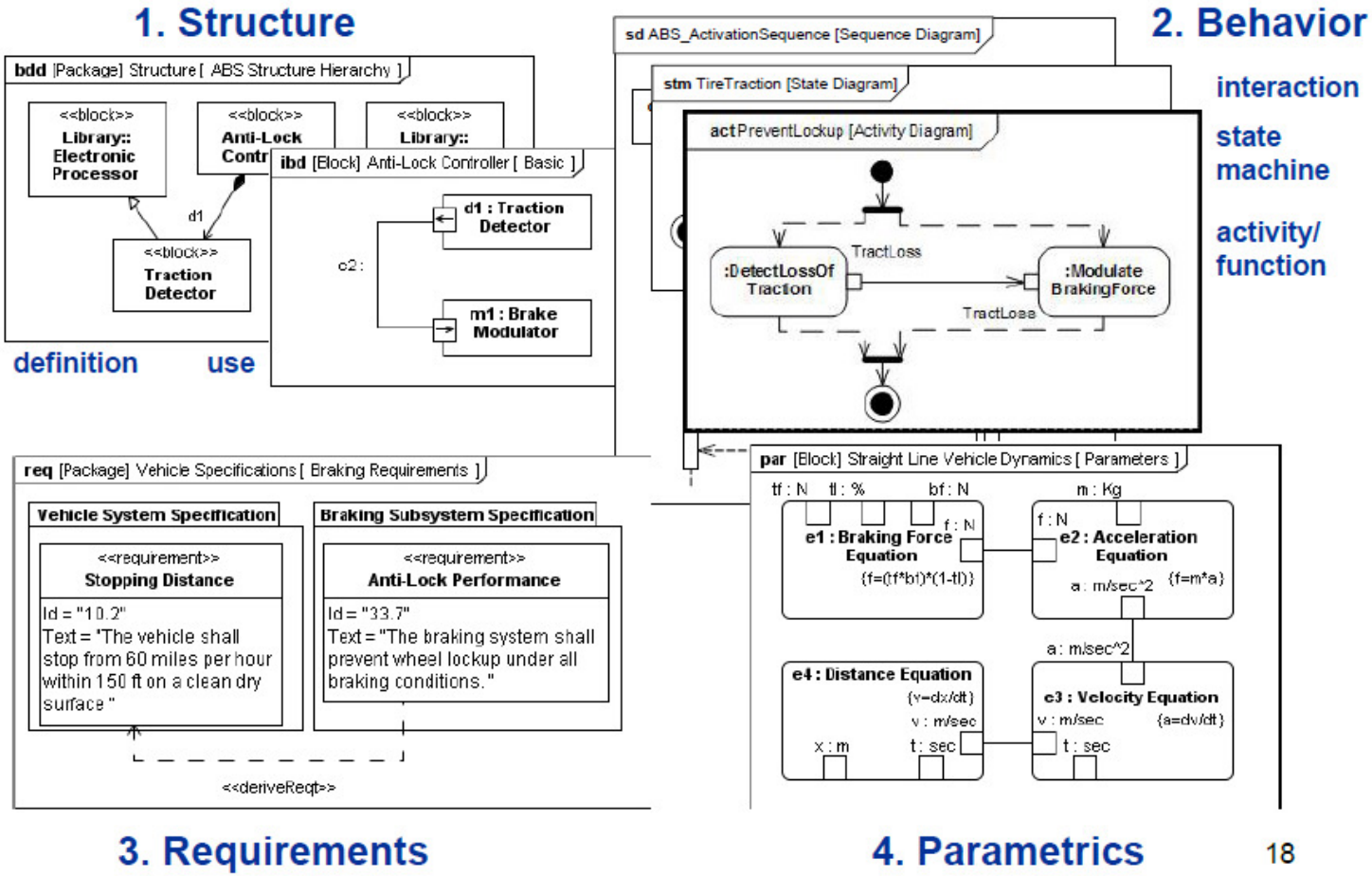
SysML

Typy diagramów SysML



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

SysML



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

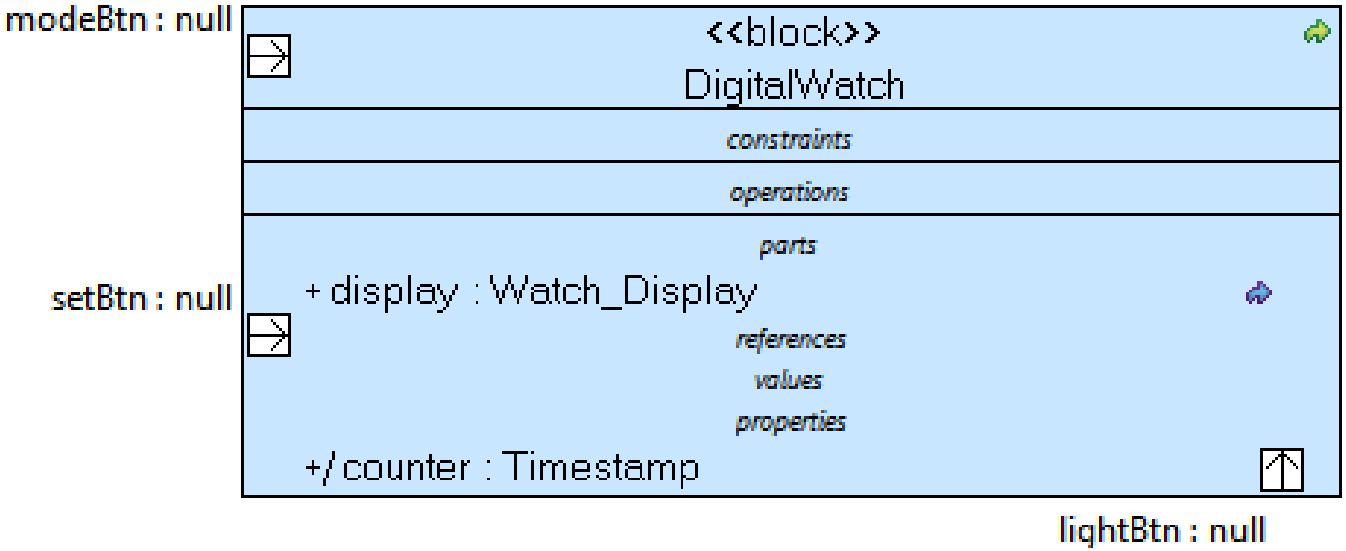
SysML

Blok – podstawowy element konstrukcyjny

- Bazuje na pojęciu Klasy UML z UML Composite Structure Diagram (dostarcza unikatowych cech, np. flow ports, value properties)
- Zapewnia jednolitą koncepcję opisu struktury elementu lub systemu
- Dowolny typ systemu/elementu – sprzęt, oprogramowanie, dane, procedury, obiektu, osoby, sygnału, wielkości fizycznej
- Przedziały (compartments) służą do opisu charakterystyki bloku – właściwości (parts, references, values, ports), operacje (operations), ograniczenia (constraints), przydziały (allocations), wymagania (requirements), zdefiniowane przez użytkownika

SysML

Reprezentacja wizualna bloku



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

SysML

Diagramy wykorzystujące bloki

- Bloki wykorzystywane są do specyfikowania hierarchii i połączeń
- **Block definition diagrams** opisuje relacje między blokami
- **Internal block diagrams** opisuje wewnętrzną strukturę bloków

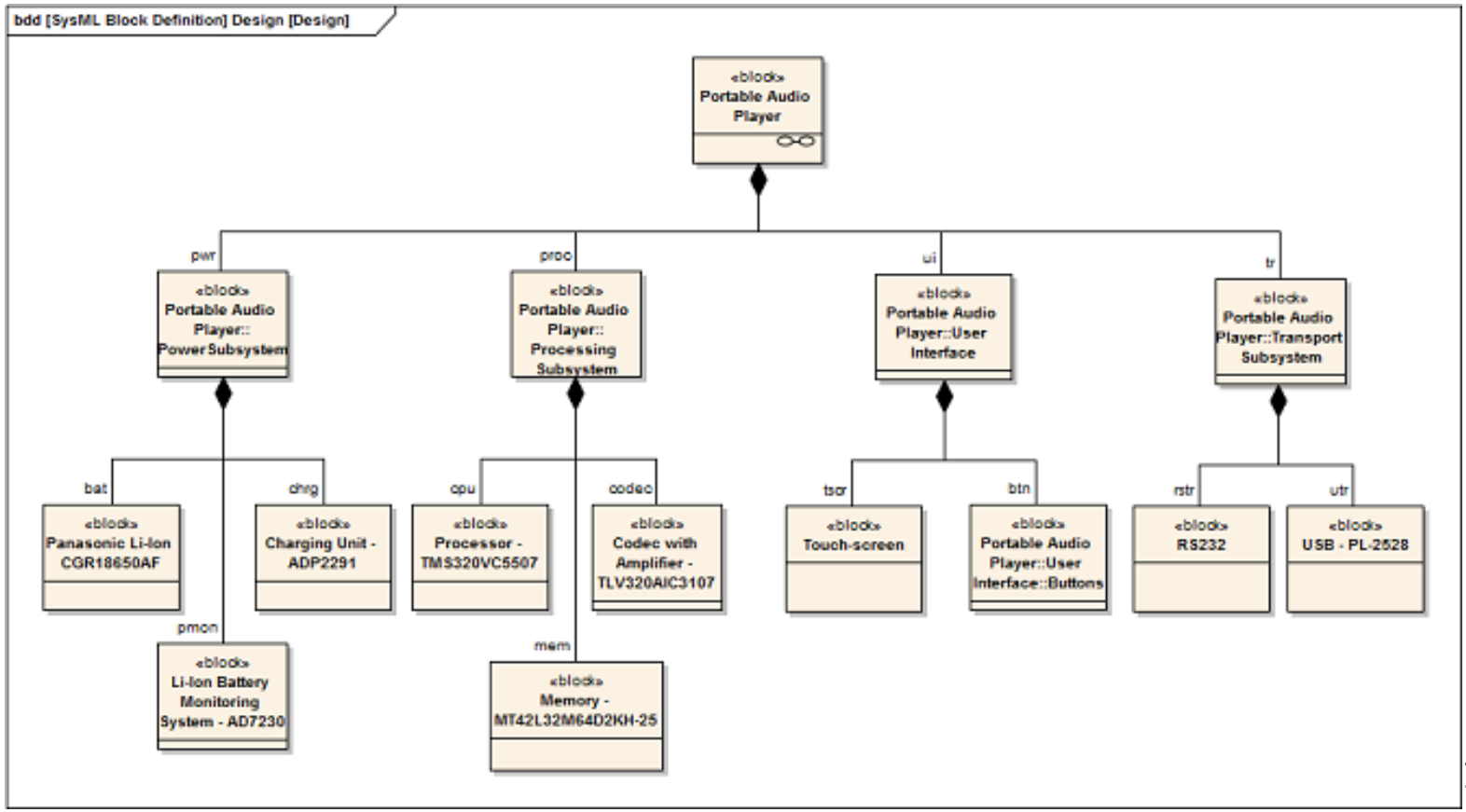
SysML

Block Definition Diagram (BDD)

- Blok w BDD odpowiada definicji typu
- Obejmuje właściwości, relacje, zależności ...
- Wykorzystywany w wielu kontekstach
- BDD nie definiuje całkowicie zależności komunikacyjnych i struktury połączeń (brak topologii)

SysML

Block Definition Diagram (BDD)



[10]

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

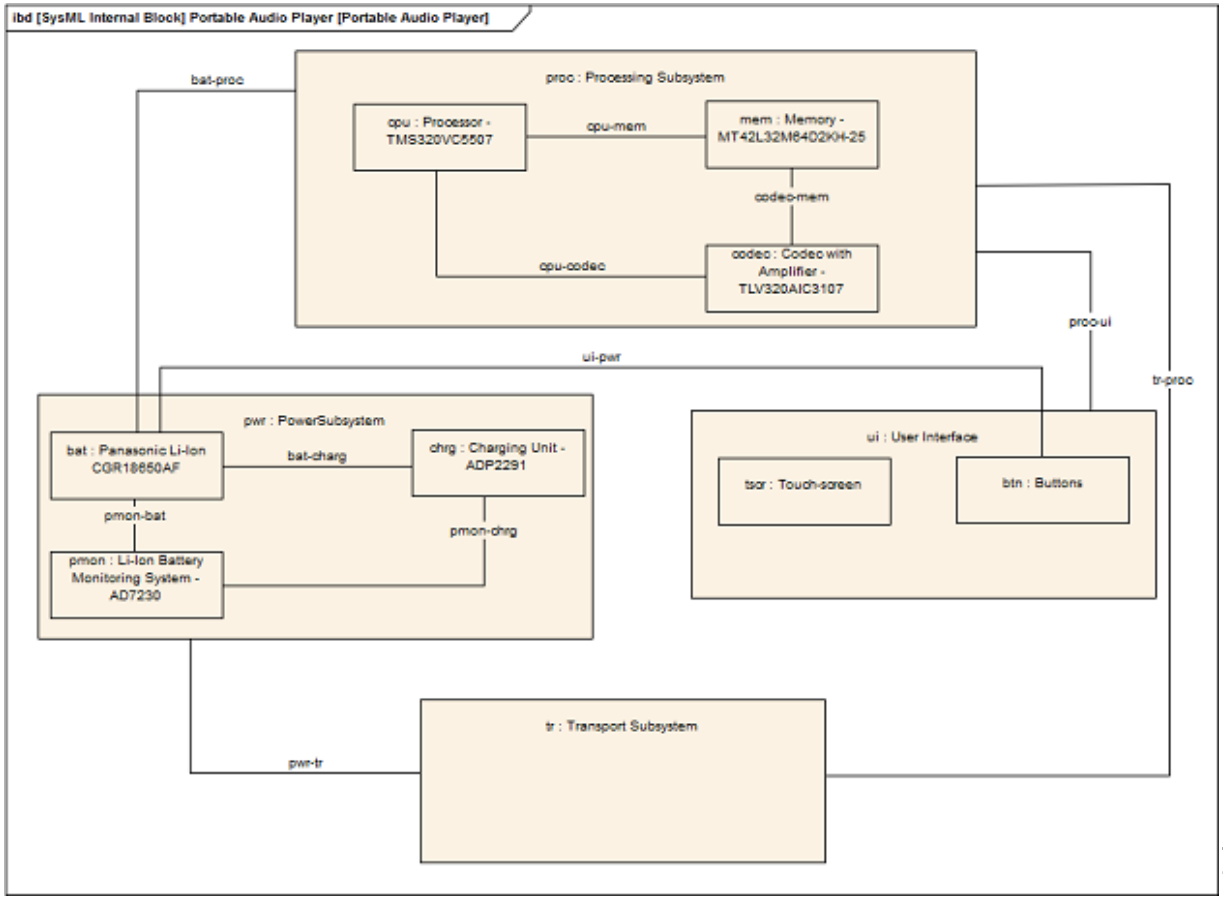
SysML

Internal Block Diagram (IBD)

- Obrazuje użycie bloków w połączeniu z innymi blokami
- Część (Part) pokazuje użycie bloku w kontekście bloku nadrzędnego
- Definiuje strukturę wewnętrzną bloku
- Komunikacja i topologia połączeń staje się jawna

SysML

Internal Block Diagram (IBD)



[11]

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

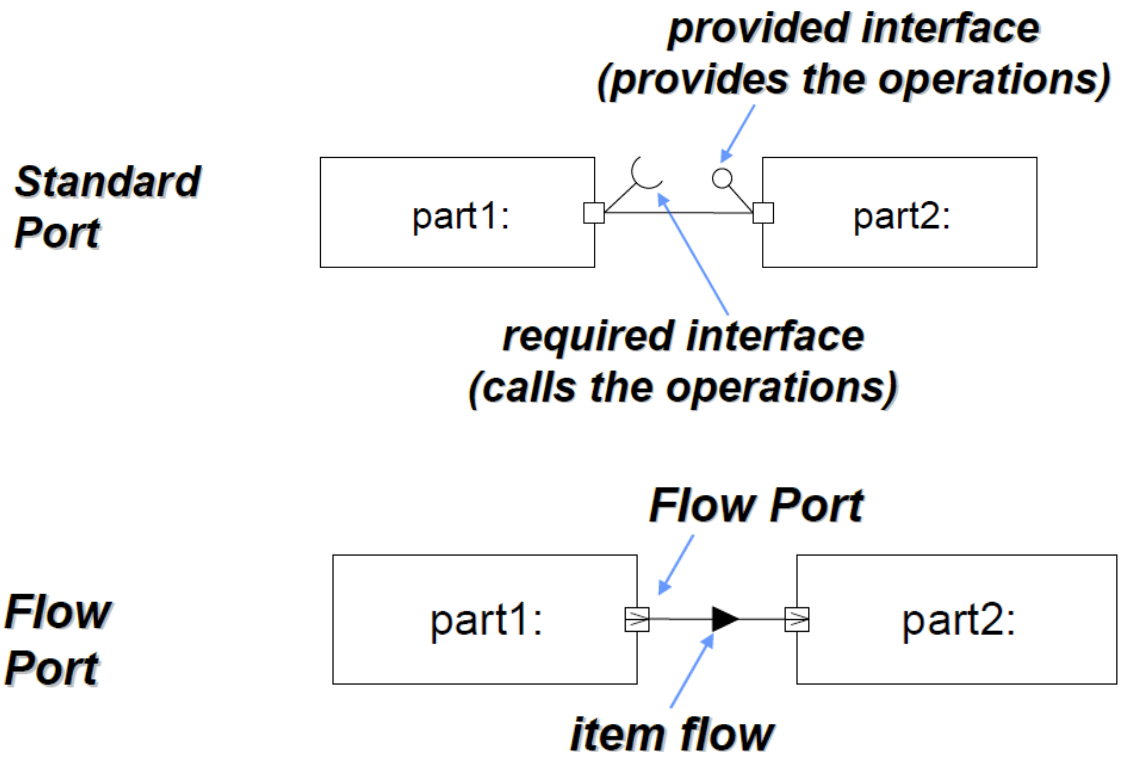
SysML

Porty SysML

- Definiuje punkty interakcji w blokach i częściach (parts)
- Integruje zachowania z elementami tworzącymi strukturę
- Składnia: portName:TypeName
- Typy portów
  - *Standard Port* (UML) – zorientowany na usługi komponentu; określa zestaw wymaganych lub dostarczonych usług i/lub sygnałów; definiowany przez interfejs UML
  - *Flow Port* – wykorzystywany przez sygnały i fizyczne przepływy; określa, co może przepływać do lub z bloku/części; definiowany przez: blok, typ lub specyfikację przepływu
- *Standard Port* i *Flow Port* wspierają różne koncepcje interfejsów
- *Flow Port* uznany za przestarzały od SysML v1.3 (*proxy, full port*)

SysML

Notacja portów



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

The End

Dziękuję za uwagę.