Komponentové systémy

definice, vývojový proces, dokumentace rozhraní

© R. Ošlejšek a B. Bühnová
Fakulta informatiky MU
oslejsek@fi.muni.cz

Motivace – Mechanické komponenty

Inspirace světem mechanických komponent

- Automobily sestavené z motoru, převodovky, kol, brzd, ...
- Počítače sestavené z procesoru, pamětí, grafické karty, monitoru, ...
- Audio sestavy, sestavené z reproduktorů, subbuferu, …

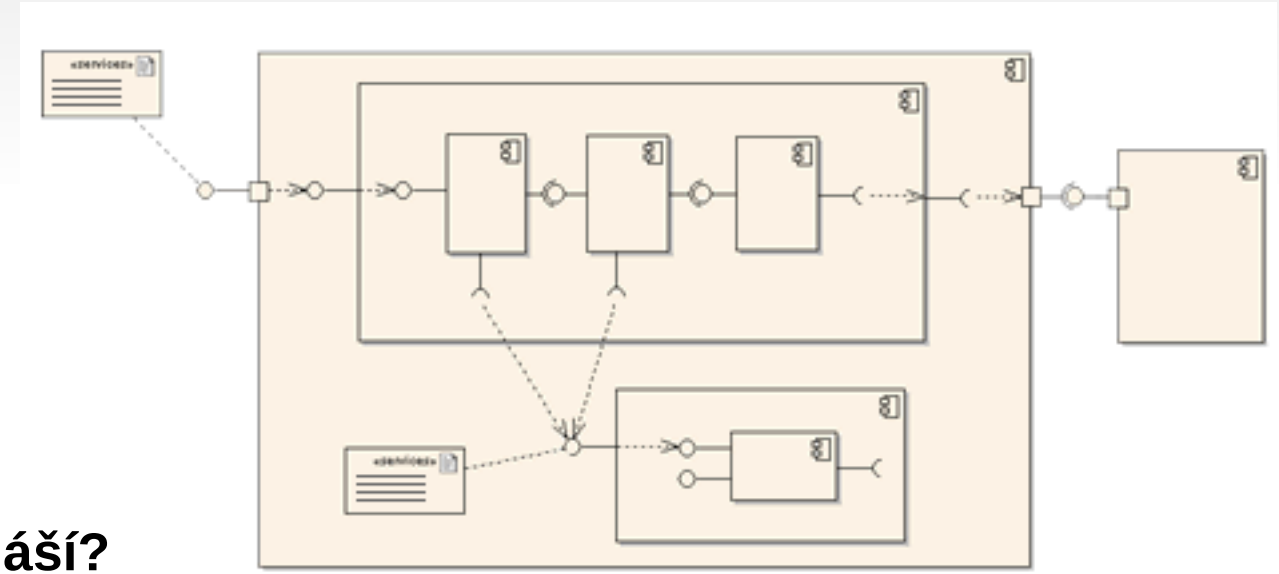
Mnoho výhod:

- Možnost rychlého sestavení celku bez znalosti vnitřního fungování jednotlivých komponent (příklad počítače) – díky standardizovaným rozhraním
- Možnost konfigurace celku výměnou jedné z komponent (např. paměti u počítače)
- … a mnoho dalších

Motivace – Softwarové komponenty

Je možné stejné výhody přenést ze světa mechanických komponent do světa software?

- Nákup softwarových komponent od různých výrobců
- Jejich propojování bez vědomí „co je uvnitř“ do funkčních systémů
- Bezpečný upgrade jednotlivých komponent



Pokud ano, co to obnáší?

Jaké to s sebou nese možné komplikace a problémy?

Existuje už dnes podpora takového vývoje?

Přehled přednášky

- Definice softwarové komponenty
- Charakteristiky SW komponent
  - Společné s mechanickými komponentami
  - Odlišné od mechanických komponent
  - Komponenta vs. objekt/třída
- Middleware pro komponentové systémy
  - Komponentové frameworky
  - Komponentové modely
- Komponentový vývoj
  - Základní business pohled
  - Komplexní vývojářský pohled
  - Vývojářské modely a role
  - Dokumentace rozhraní komponent
    - Quality of Service (QoS)
    - Service Level Agreement (SLA)

Definice Softwarové komponenty

A reusable software component is a logically cohesive, loosely coupled module that denotes a single abstraction. [Booch 1987]

Software components are defined as prefabricated, pretested, self-contained, reusable software modules – bundles of data and procedures – that perform specific functions. [Meta Group 1994]

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. [Heineman & Councill 2001]

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. [Szyperski 2002]

A component represents a modular piece of a logical or physical system whose externally visible behaviour can be described much more concisely than its implementation. Components do not depend directly on other components but on interfaces that components support. [UML 2.0 Standard 2004]

Vazba na softwarové architektury

Komponentové systémy (Component-Based Systems) jsou konkrétní realizací softwarových architektur, kdy:

1. **Moduly** = black-box komponenty komunikující výhradně skrz publikovaná rozhraní

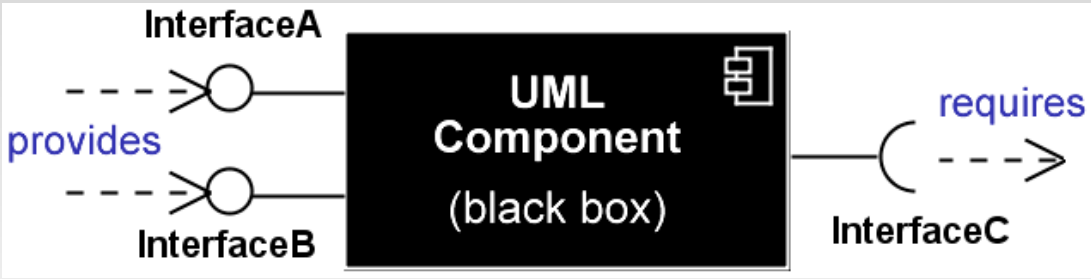
2. **Konektory** = spoje a kanály zajišťující komunikaci a spolupráci zkompilovaných modulů (komponent) implementovaných často odlišnými prostředky a nezávisle na sobě

3. **Nasazení** = mapování modulů a konektorů na hardwarové (nebo softwarové) zdroje

- Např. síťové linky, fyzické zdroje a servery v případě hardwarových zdrojů
- Např. komponentový framework či middleware v případě softwarových zdrojů

Charakteristiky – Společné s mech. komponentami

- Zapouzdření
- Rozhraní a služby na nich
  - Jedinný přístupový bod k funkcionalitě komponent



- Anonymita klienta i serveru
  - Komponenta při obsluze/vyslání požadavku nezná iniciátora/příjemce požadavku
- Připravenost k okamžitému použití
  - Netřeba kompilovat po zapojení do systému
- Black-box znovupoužitelnost
- Snadná vyměnitelnost
  - Systém je možné jednoduše konfigurovat výměnou stávajících komponent
- Jazyková nezávislost
  - Každá komponenta může být implementována v jiném jazyce

Charakteristiky – Odlišné od mech. komponent

- Hierarchická struktura (vnitřních komponent)
  - I komponenta složená z podkomponent může být skládána s dalšími komponentami – vzniká tak hierarchická architektura
- Instanciovatelnost
  - Vznik „nových komponent“ za běhu
- Dynamičnost
  - Možnost změny komunikačního partnera za běhu
- Vysoká míra složitosti a variability
  - Tudíž i obtížná standardizovatelnost
- Vliv (předem neznámých) fyzických zdrojů
  - Extra-funkční parametry komponenty nelze odhadnout v době implementace
- Silná závislost na vnitřním stavu
  - Odlišné chování v závislosti na historii
- Závislost na běhovém prostředí - middlewaru

Čím se liší komponenta od objektu/třídy?

Komponenta (vs. objekt/třída):

- Spustitelná běhová jednotka
- Definovaná rozhraními (IDL)
- Typicky složena z více tříd/objektů (hierarchicky)
- Bez viditelného zdrojového kódu (black-box/grey-box)
- Vyvinuta odděleně od ostatních komponent (pro znovupoužitelnost)
- Kompilace předchází zasazení do kontextu
- Závislost na běhovém prostředí (middlewaru)

Black box view = poskytovaná a požadovaná rozhraní, (neúplná) textová dokumentace

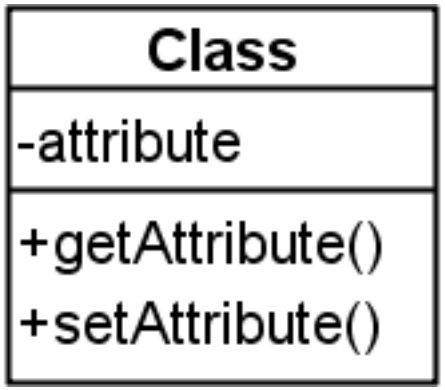
Grey box view = poskytované a požadované služby, architektura vnitřních komponent a jejich vnitřních komponent (až na úroveň primitivních komponent, tvořených přímo kódem)

Čím se liší komponenta od objektu/třídy?

Objekt/třída (vs. komponenta):

- Návrhová jednotka
- S viditelným zdrojovým kódem (white-box)
- Typicky navržena pro konkrétní systém
- Zasazení do kontextu (ostatních tříd) předchází kompilaci

White box view = zdrojový kód



Příklady zajímavých problémů

Instanciovatelnost a replikace komponent

- Kam fyzicky nasadit nově vzniklou komponentu?
- Kam ji umístit v hierarchii komponent?

Interoperabilita

- Vzájemná kompatibilita komunikujících komponent
- Kompatibilita typů v signatuře volajícího (např. int) a volaného (např. real)
- Generování a použití adaptorů a wrapperů

Nahraditelnost komponenty

- Otázka korektnosti nahrazení stávající komponenty novou (rekonfigurace systému)
- Nová komponenta by neměla
  - Požadovat více
  - Nabízet méně
- Nejen mezi implementacemi, ale i nahrazení specifikace implementací (při hledání komponenty realizující návrh)

Middleware pro komponentové systémy

Middleware obecně

Middleware is a name for the set of software that sits between various operating systems and a higher, distributed programming platform. [Szyperski 2002]

- Např. J2EE nebo .NET

Komponentové frameworky

A component framework is a software entity that supports components conforming to a certain standards and allows instances of these components to be plugged into the component framework. The component framework establishes evnironmental conditions for the component instances and regulates the interaction between component instances. [Szyperski 2002]

- Např. EJB pro J2EE, nebo COM/COM+ pro .NET
- Konkrétní typ middlewaru pro komponentové systémy

Komponentové modely

A component model defines specific representation, interaction, composition, and other standards for software components. [Heineman & Councill 2001]

- Např. EJB pro J2EE, nebo COM/COM+ pro .NET
- Na rozdíl od komponentových frameworků, které implementují běhové prostředí, zde jde spíše o sadu standardů.

Komponentové modely

Základní představitelé:

- Komerční komponentové modely
  - CCM/CORBA, EJB/J2EE, COM/.NET
  - Spjaté s komponentovými frameworky pro praktickou realizaci komp. systémů
- Akademické komponentové modely
  - Fractal, KobrA, PCM, SOFA, Darwin, Wright, Koala, ACME, …
  - Spjaté nejčastěji s analytickými nástroji pro ověření korektnosti či predikci kvality komponentových systémů

Existující modely mají různé pohledy na následující:

- Co se rozumí komponentou
- Jak jsou popsána rozhraní/služby
- Jak jsou komponenty skládány dohromady

Co se rozumí komponentou

Run-time vs. design-time jednotka

- Run-time – implementovaný modul, pro realizaci konkrétní funkcionality [COM/.NET, Fractal]
- Design-time – návrhový prvek, pro ohodnocení kvality navrhované komponenty [KobrA, SOFA, PCM]

Dynamický vs. statický prvek

- Dynamický – možnost vytváření instancí za běhu [EJB, CCM, Darwin]
- Statický – analogie mechanických komponent, komponenty pevně dané [Wright, SOFA]

Stavový vs. bezestavový prvek

- Stavový – komponenta má vnitřní stav (paměť), analogie atributů u objektu [CCM, EJB, KobrA]
- Bezestavový – komponenta nemá vnitřní paměť, její reakce na příchozí požadavky nejsou ovlivněny historií [SOFA, Wright]

Jak jsou popsána rozhraní/služby

Signatury vs. protokoly vs. vstup/výstupní podmínky vs. popis chování

- Signatury – název, typy vstupních argumentů, výstupní typ
  [EJB, COM/.NET, CCM]
- Protokoly – posloupnosti validních posloupností volání služeb
  [PCM, SOFA, Wright]
- Vstupní/výstupní podmínky – podmínky platící před voláním a po provedení služby
  [KobrA]
- Popis chování – strukturovaný (zjednodušený) popis implementace služby
  [SOFA, PCM]

Tomuto tématu bude věnována zvláštní pozornost dále

Jak jsou komponenty skládány dohromady

Synchronně vs. asynchronně

- Synchronně – volání služeb, volající čeká než je volání zpracováno [Darwin, SOFA]
- Asynchronně – zasílání zpráv, volající nečeká než je zpráva zpracována [EJB, COM/.NET, CCM]

Jednoúrovňové vs. víceúrovňové skládání

- Jednoúrovňové – složení komponent tvoří přímo systém [EJB, COM/.NET, CCM]
- Víceúrovňové (hierarchické) – složení komponent tvoří opět komponentu, která může být dále skládána s jinými komponentami [Fractal, SOFA, Koala, ACME]

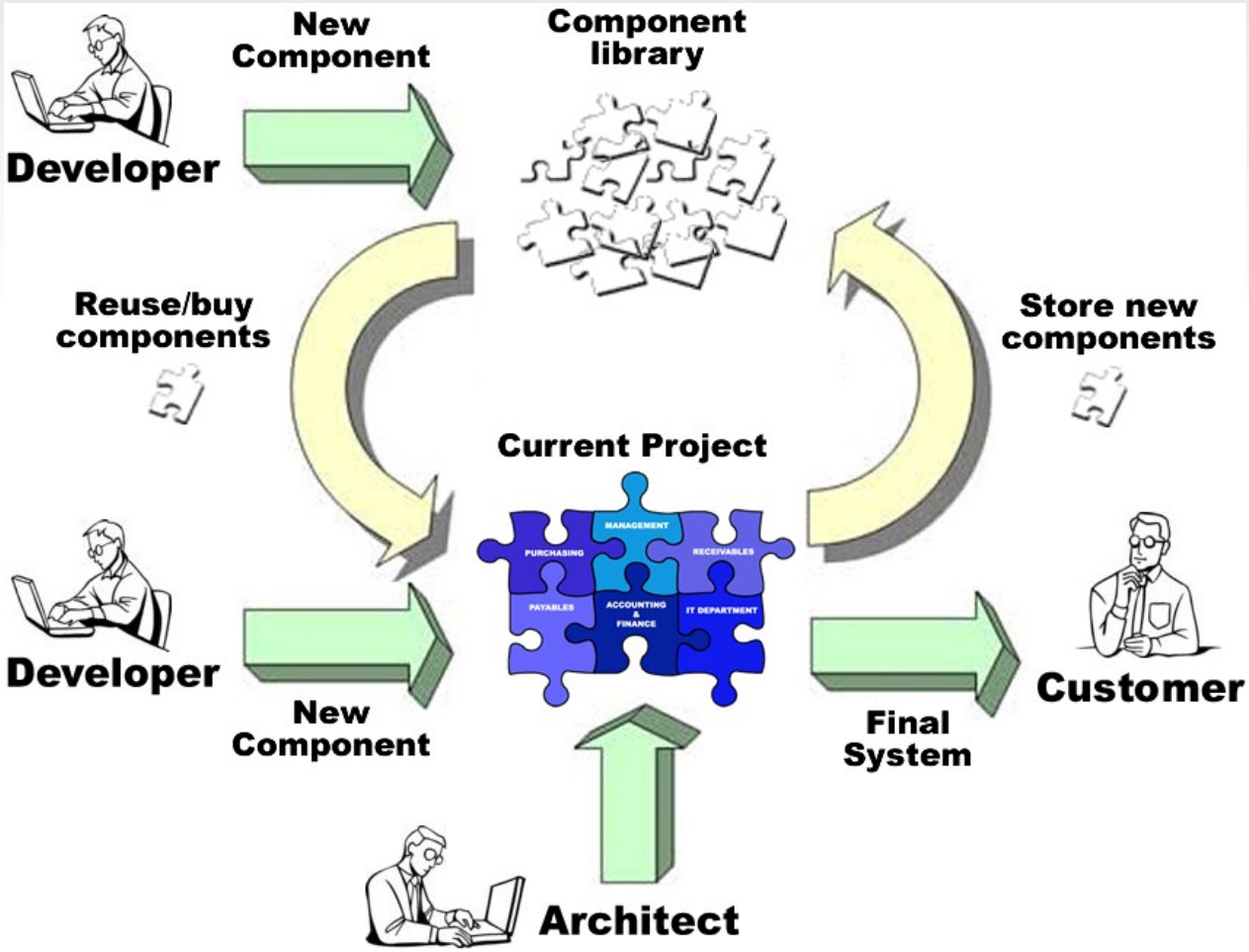
Jednoduché vs. násobné propojení mezi komponentami

- Jednoduché – každé rozhraní je napojeno na maximálně jednu protistranu [SOFA]
- Násobné – jedno poskytované rozhraní může být napojeno na více požadovaných rozhraní, a naopak [EJB, Fractal]

Komponentový vývoj

Základní business pohled:

- Vzájemně nezávislé fáze vývoje komponent a sestavování systému.



Výhody komponentového vývoje

Knihovny komponent

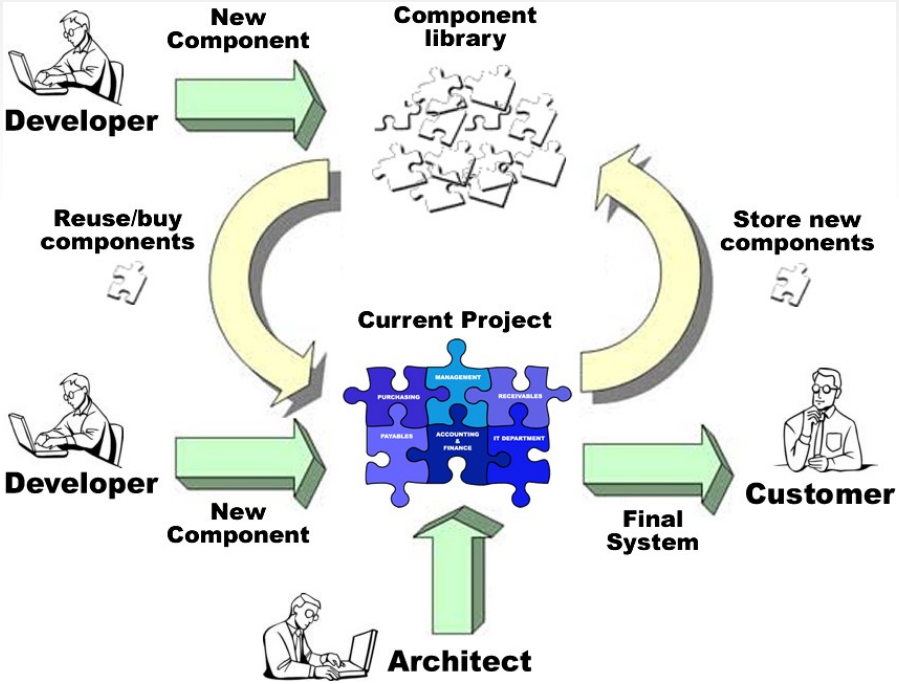
- Urychlení vývoje
- Snížení rizika (ověřené komponenty)

Znovupoužitelnost komponent

- Nižší náklady
- Vyšší kvalita

Udržovatelnost

- Srozumitelnost
- Konfigurovatelnost
- Jazyková nezávislost
- Flexibilita, snadnost výměny



Komponentový vývoj

Komplexní vývojářský pohled:

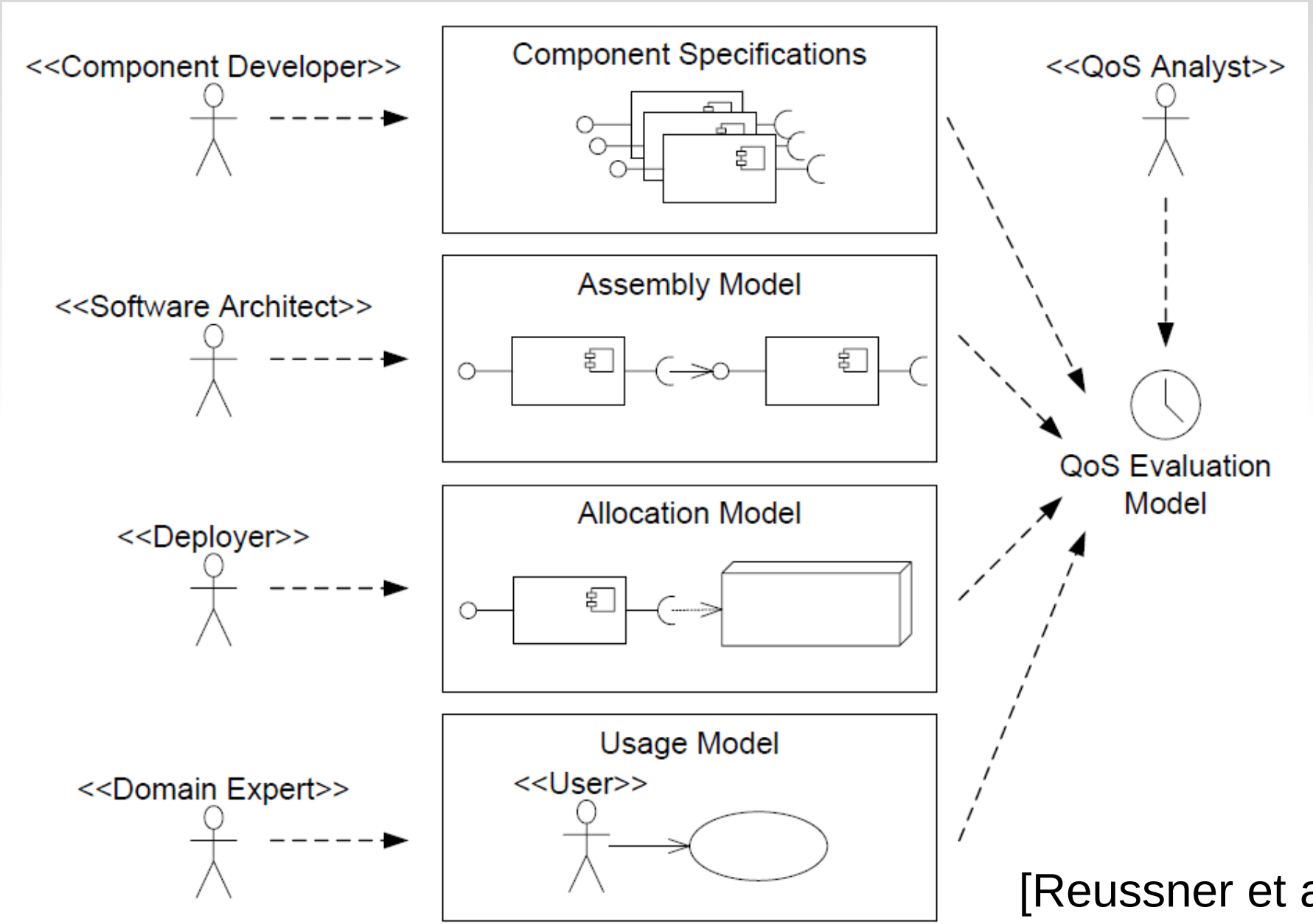
- Fáze vývoje komponent
- Fáze sestavení systému
- Fáze nasazení systému na fyzické/logické zdroje
- Fáze ohodnocení kvality systému

Jednotlivé fáze vývoje probíhají **relativně nezávisle**

- Vývojář komponenty neví, do jakého kontextu bude komponenta architektem zasazena, ani na jaké zdroje bude nasazena
- Architekt sestavující systém nezná implementaci komponent ani jejich budoucí nasazení

Vzájemná komunikace probíhá jen skrz modely dokumentující jejich výstupy → vznik nezávislých **vývojových modelů**, pod zodpovědností oddělených **vývojářských rolí**

Vývojářské modely a role



[Reussner et al.]

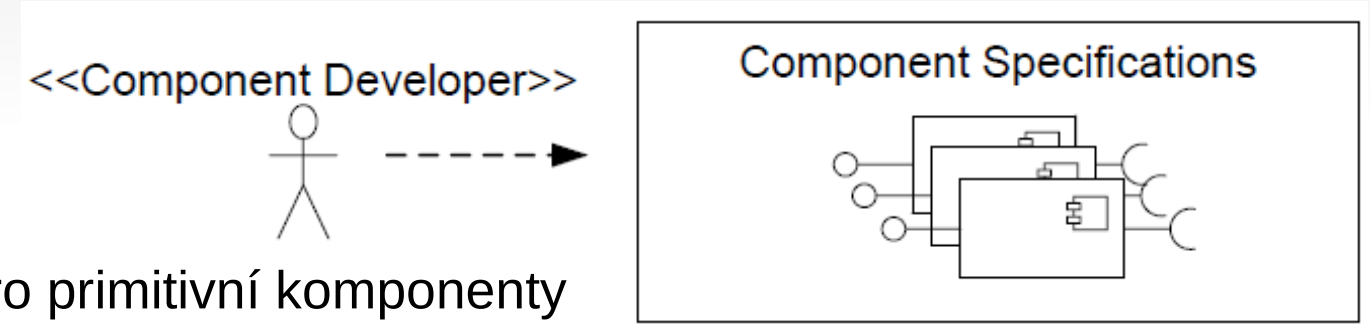
Komponentový vývojář (Component Developer)

Role: Komponentový vývojář

- Implementace a dokumentace jednotlivých komponent (zejména jejich rozhraní a omezení na komunikaci skrz ně)

Model: Specifikace komponenty

- Popis vnitřní struktury a rozhranní vyvinutých komponent



UML:

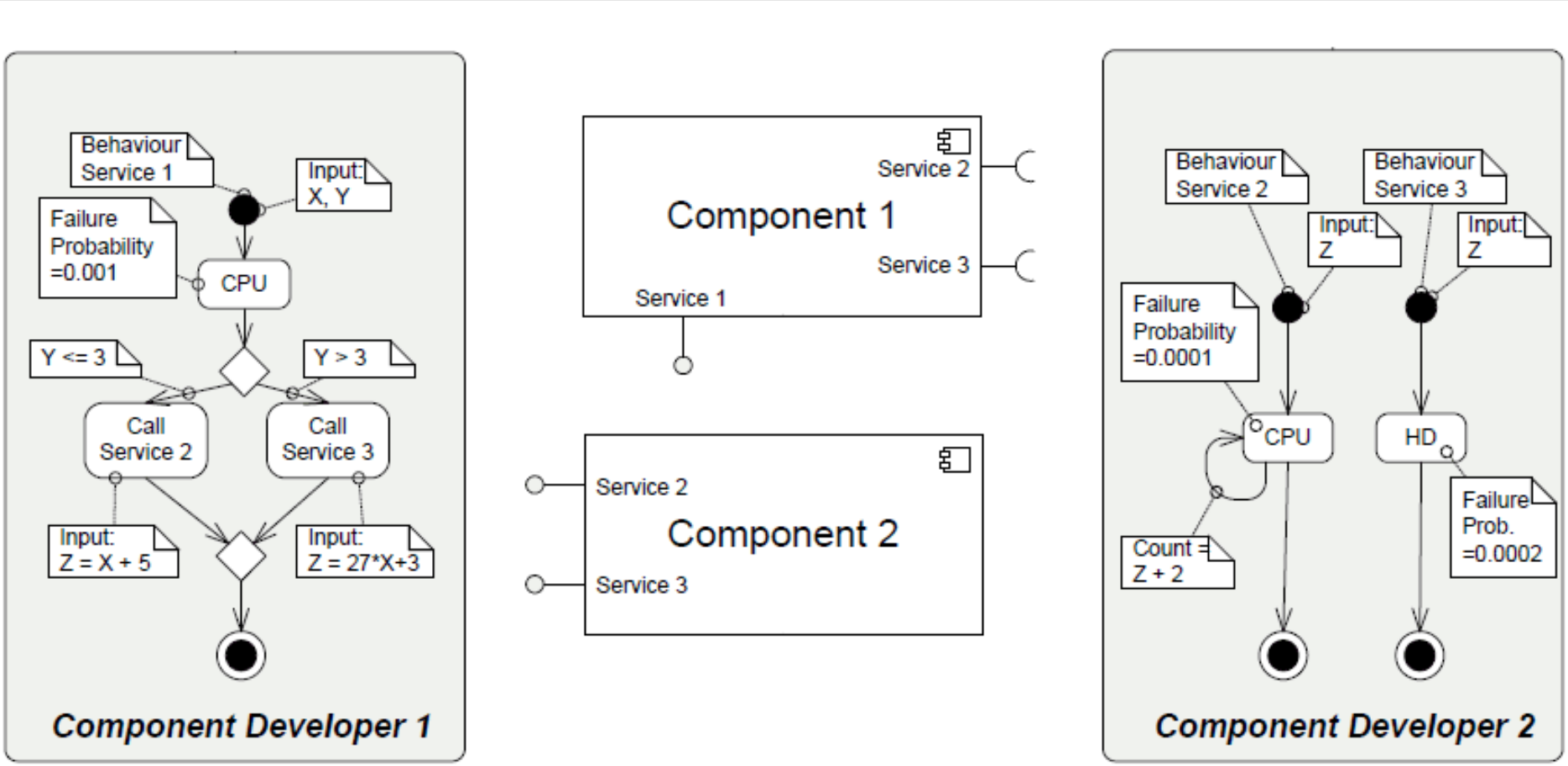
- Diagram tříd/objektů pro primitivní komponenty
- Komponentový diagram pro složené komponenty

V obou případech

- Interakční diagramy (interakce mezi třídami/komponentami)
- Diagram aktivit (chování služby nabízené na rozhraní)
- Stavové diagramy (stavy komponenty)

Příklad – Specifikace komponenty

Příklad specifikace komponenty z pohledu chování služeb nabízených na rozhraní (anotovaný diagram aktivit):



Softwarový architekt (Software Architect)

Role: Softwarový architekt

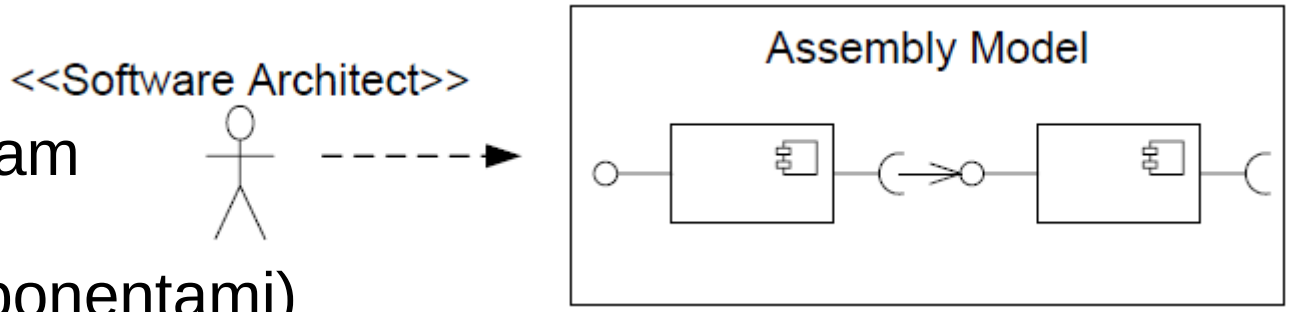
- Sestavení komponent do celkového systému na základě popisu jejich rozhraní
- Specifikace vnějších rozhraní systému (pro komunikaci s uživateli i externími systémy) delegací vnitřních rozhraní
- Dokumentace použitých konektorů

Model: Model sestavení systému

- Statická struktura a dynamický procesní model architektury

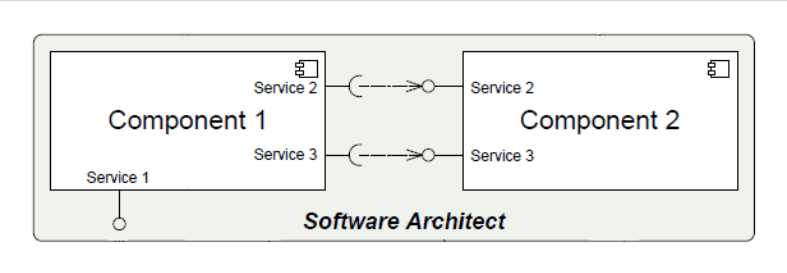
UML:

- Komponentový diagram
- Sekvenční diagramy (interakce mezi komponentami)

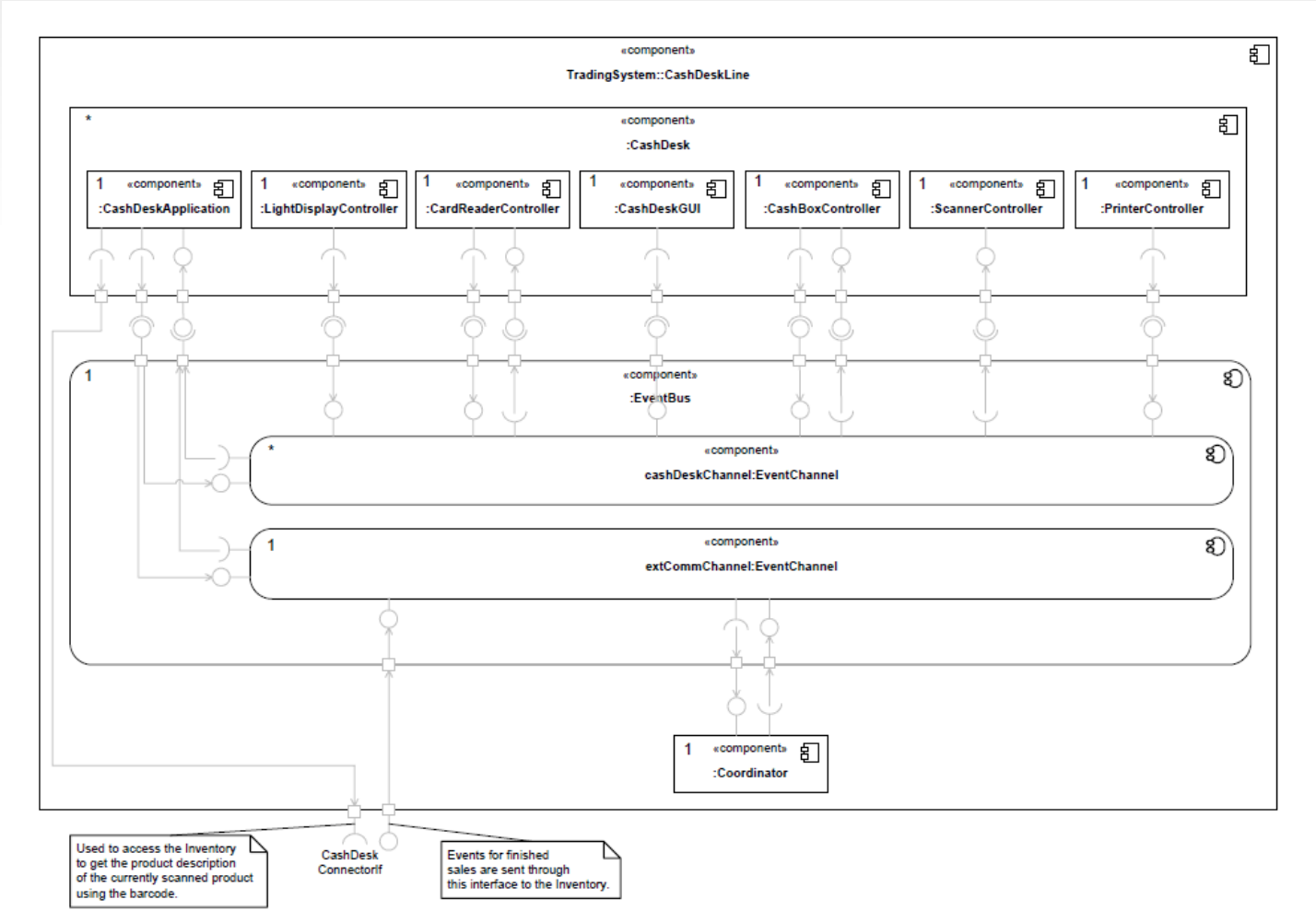


Příklad – Model sestavení systému

Jednoduchý příklad:



Složitější příklad:



Expert pro nasazení systému (Deployer)

Role: Expert pro nasazení systému

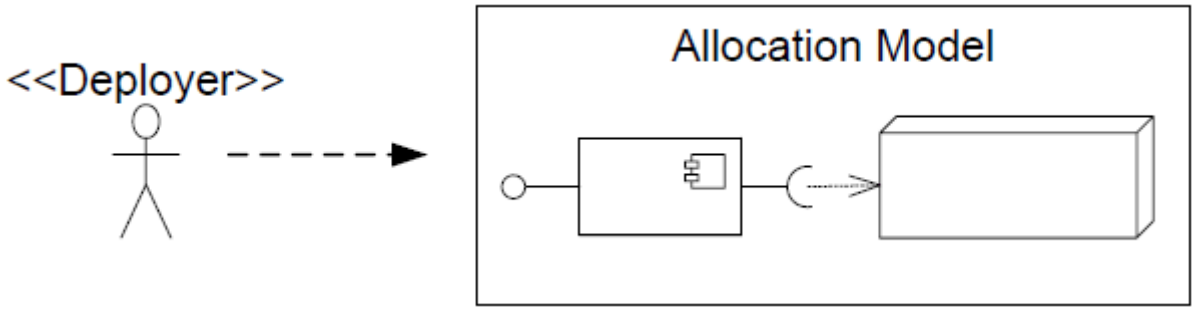
- Volba fyzických zdrojů (včetně jejich parametrů)
- Návrh fyzické architektury systému
- Mapování softwarových komponent na zvolené zdroje

Model: Model nasazení systému

- Statický model mapování systémových komponent na rozvržené zdroje

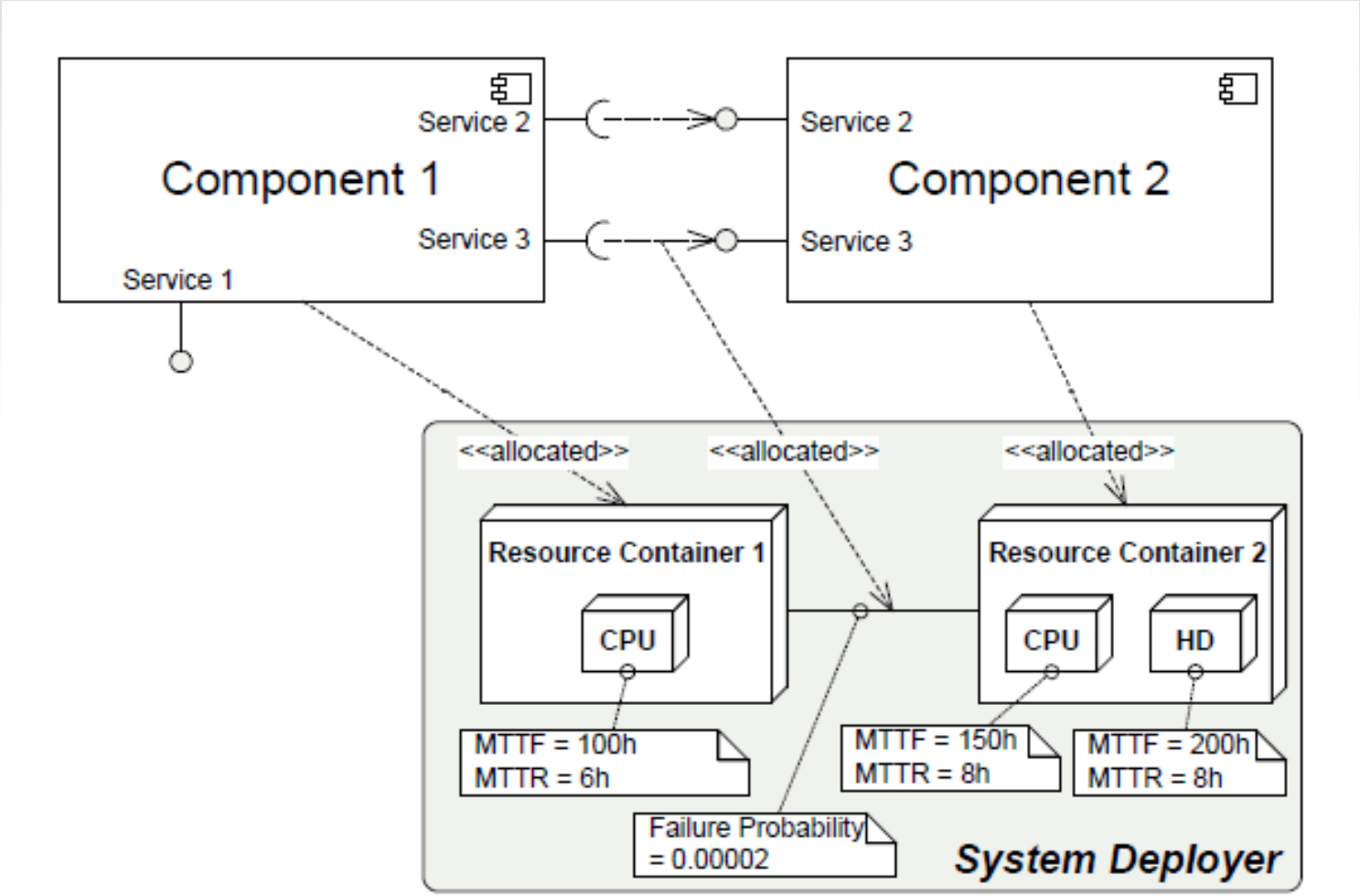
UML:

- Diagram nasazení



Příklad – Model nasazení systému

Příklad:



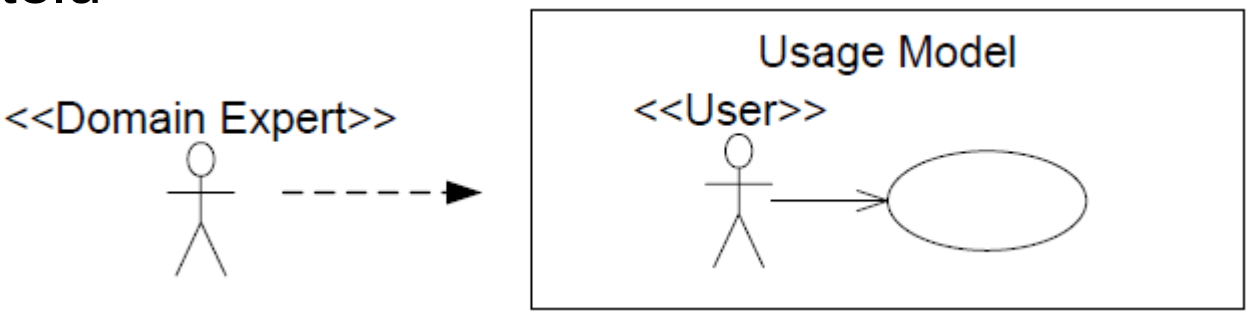
Doménový expert (Domain Expert)

Role: Doménový expert

- Vykomunikování očekávaných scénářů používání systému
- Naspecifikování sekvencí volání systému, očekávaných argumentů těchto volání
- Odhadnutí očekávaného počtu souběžných uživatelů

Model: Model používání systému

- Popis sekvencí volání systémových služeb uživatelem
- Informace o počtu uživatelů

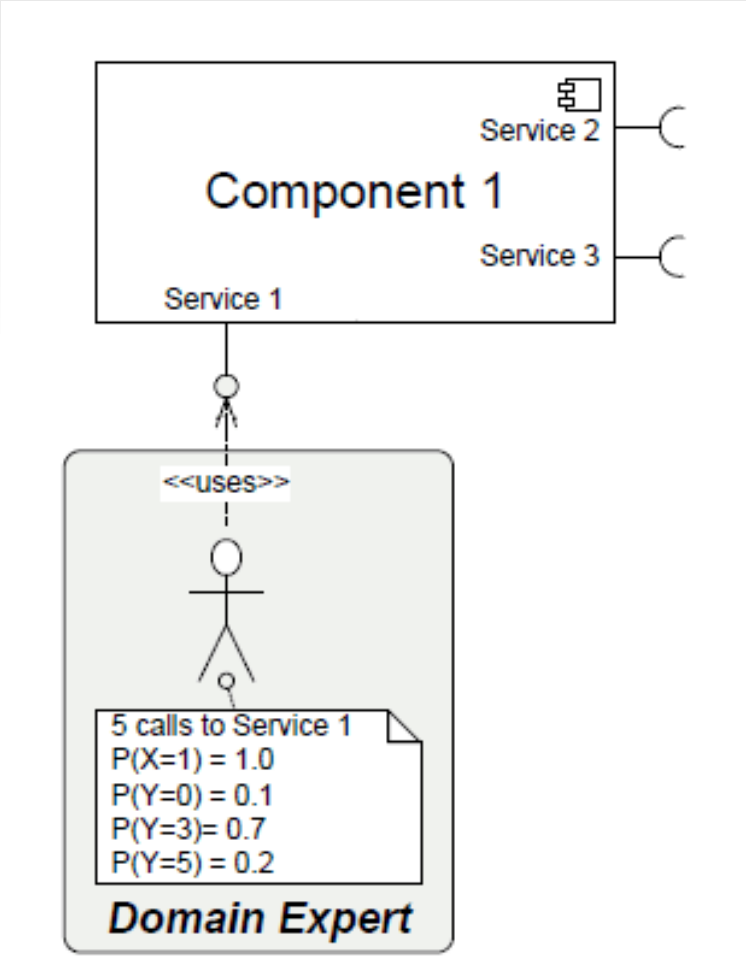


UML:

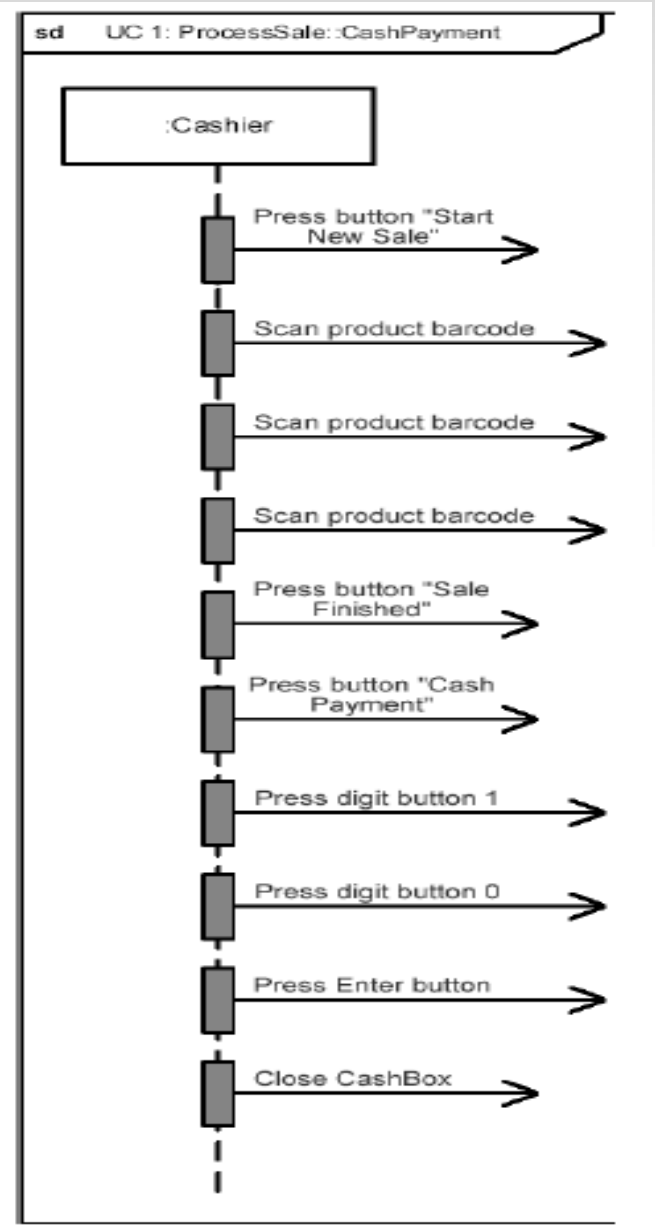
- Sekvenční diagram (interakce mezi aktorem a komponentami na rozhraní systému)

Příklad – Model používání systému

Příklad 1:



Příklad 2:



QoS analytik (QoS Analyst)

Role: QoS analytik

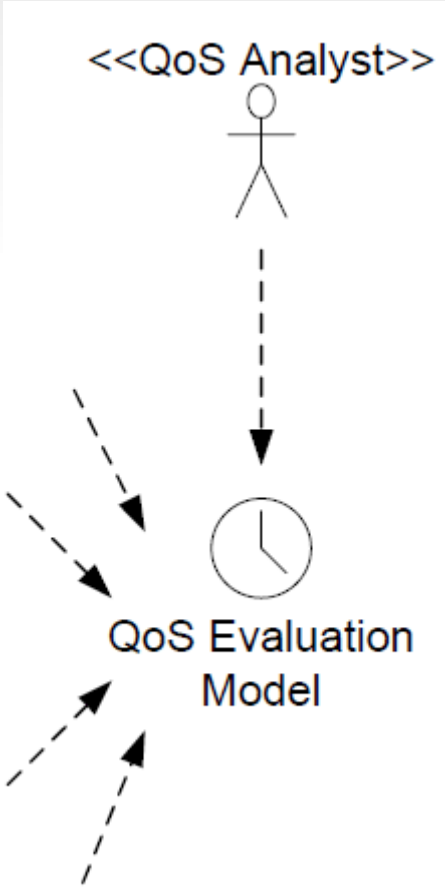
- Volba kvalitativních atributů k ohodnocení
- Provedení analýzy – ohodnocení jednotlivých kvalitativních kritérií celkové architektury na základě modelů dodaných všemi rolemi
- Interpretace výsledků – impuls k modifikaci architektury

Model: Model ohodnocení kvality služeb

- Formální model obsahující informace z předchozích modelů
- Částečně generovaný automaticky
- Podklad pro automatizovanou analýzu kvality systému

Prostředky:

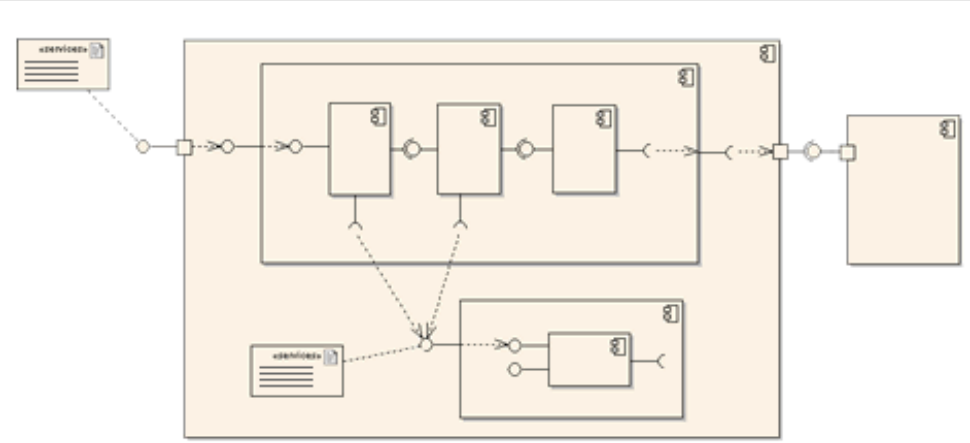
- Finite Automata (FA)
- Markov Chains (MC)
- Layered Queuing Networks (LQN)



Dokumentace rozhraní komponent

Význam:

- Anotace rozhraní komponent dostatečným množstvím informací pro jejich použití



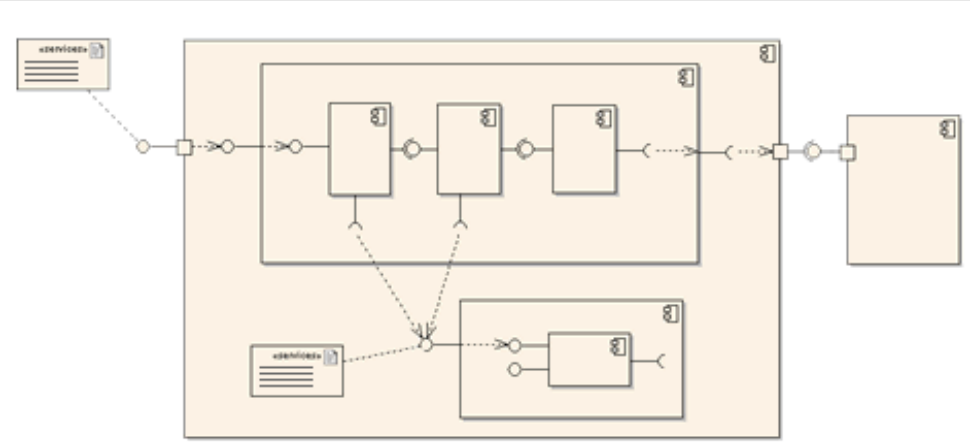
Typy informací:

- Signatury služeb – název, typy vstupních argumentů, výstupní typ
- Protokoly doporučeného použití služeb – UML sekvenční diagramy
- Vstupní a výstupní podmínky – formální/neformální popis
- Strukturovaný popis chování služeb – UML aktivity diagramy
- Garantované kvality služeb – Quality of Service (QoS), Service Level Agreement (SLA)

Dokumentace rozhraní komponent

Význam:

- Anotace rozhraní komponent dostatečným množstvím informací pro jejich použití



Typy informací:

- Signatury služeb – název, typy vstupních argumentů, výstupní typ
- Protokoly doporučeného použití služeb – UML sekvenční diagramy
- Vstupní a výstupní podmínky – formální/neformální popis
- Strukturovaný popis chování služeb – UML aktivity diagramy
- Garantované kvality služeb – Quality of Service (QoS), Service Level Agreement (SLA)

Garantované kvality služeb

Týkají se **nefunkčních (extra-funkčních) kvalitativních atributů**, jako např:

- **Výkonnost (performance)** – propustnost, doba odezvy, efektivita využití zdrojů
- **Spolehlivost (reliability)** – bezchybný provoz, dostupnost, robustnost, zotavitelnost
- **Bezpečnost (security)** – důvěrnost, integrita, dostupnost

Způsob specifikace

- Minimální a maximální hodnota
- Průměrná (average) a střední (mean) hodnota
- Pravděpodobnostní rozložení možných hodnot

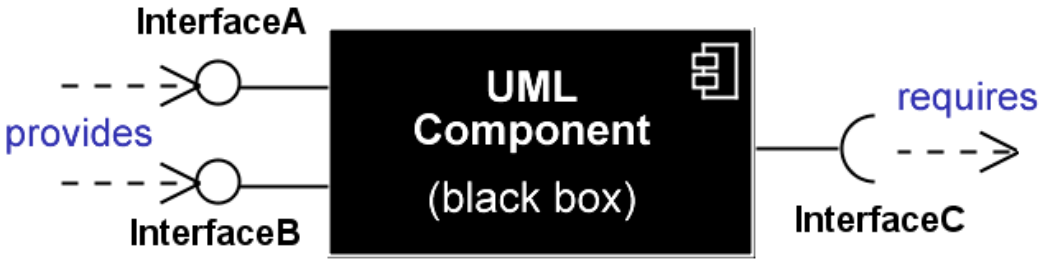
Quality of Service (QoS)

Garantovaná kvalita služby:

- Anotace rozhraní s informací o kvalitě každé služby
- Vyjádřeno pro každý sledovaný kvalitativní atribut

Naivní přístup: anotace konkrétní `service()` ve stylu…

- Garantovaná doba odezvy max. 10 ms v 99.9% případů.
- Spolehlivost (bezchybný provoz) 99.9% v rámci každého volání služby.



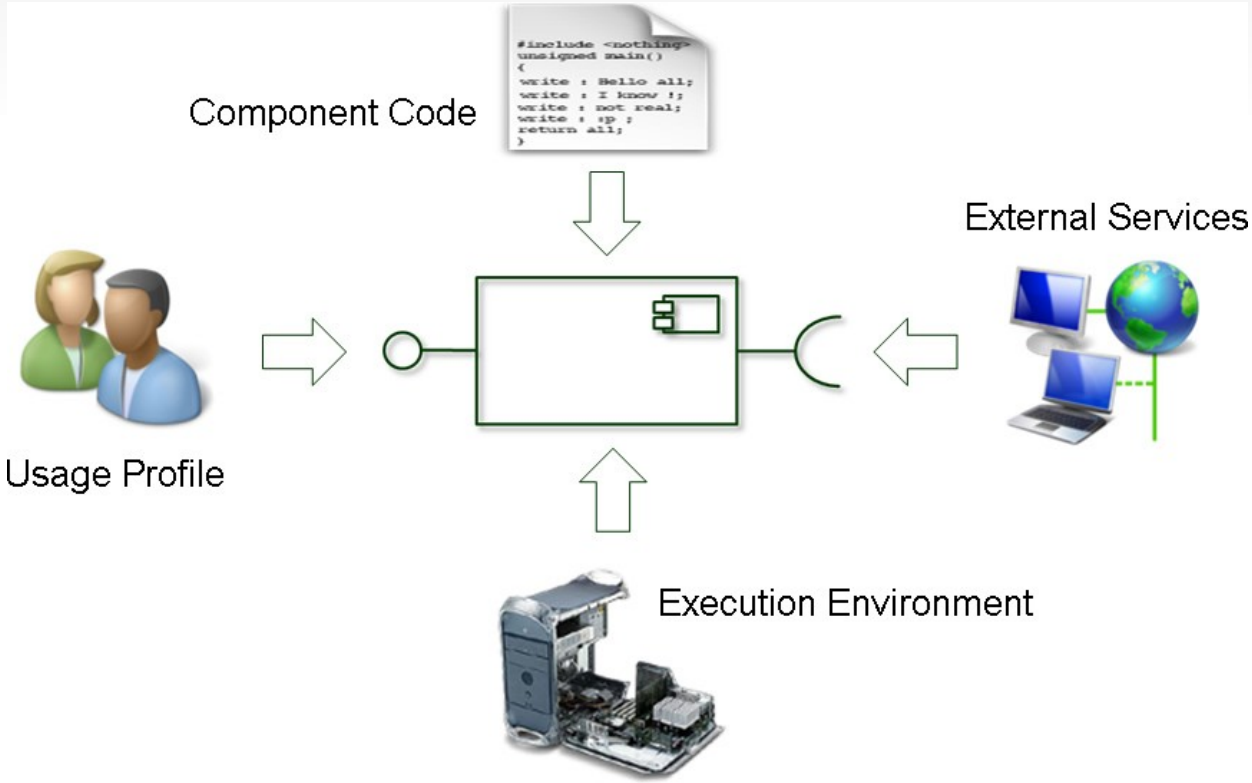
Problém:

- Těžko mohu garantovat konkrétní čísla, když předem nevím, jaké kvality mohu předpokládat od požadovaných služeb (jak budou rychlé/spolehlivé/…)

Kontext komponenty/služby

Garantovatelné kvality služby závisí na:

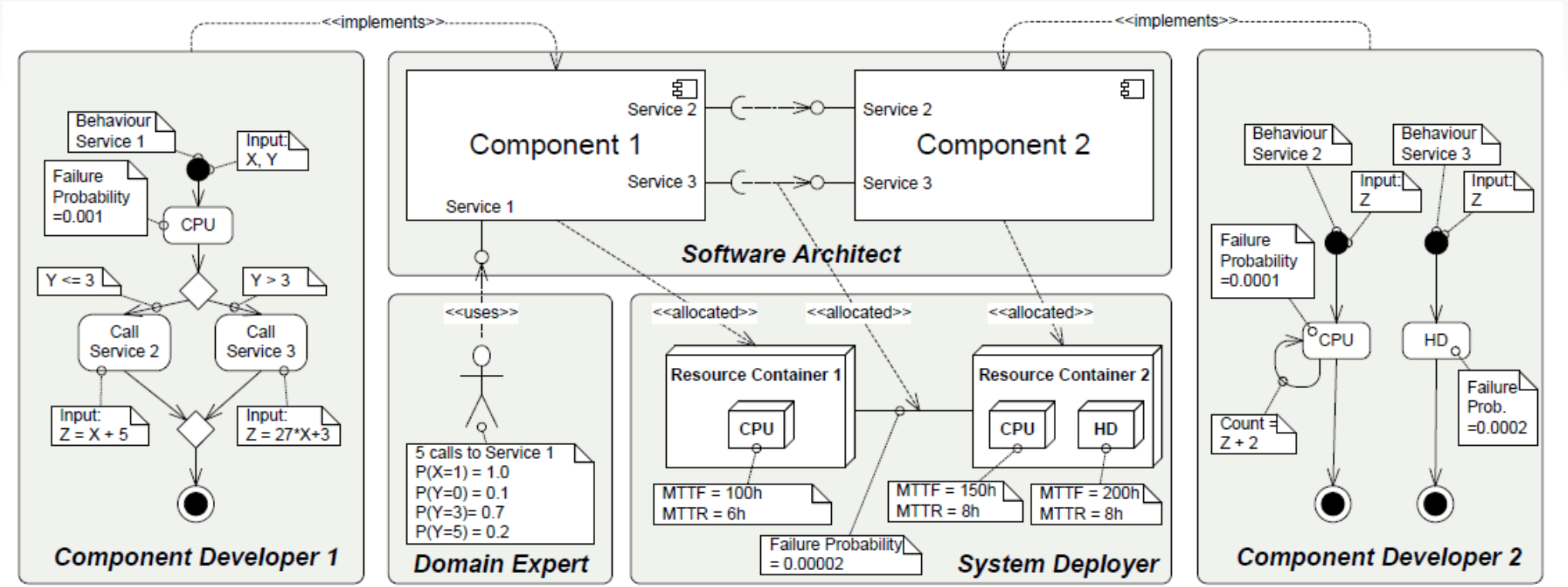
- Implementaci služby
- Kvalitě požadovaných služeb
- Zdrojích, na kterých je komponenta nasazena
- Způsobu použití služby



Příklad kontextu komponenty/služby

Kontext služby `Service1()`:

- Implementaci služby (Component Developer 1)
- Požadované služby (Software Architect, Component Developer 2)
- Zdroje kde je služba nasazena (System Deployer)
- Použití služby (Domain Expert)



Typy QoS specifikace

Kvality z pohledu KLIENTA SLUŽBY

Pokud je celý kontext známý – typicky v případě služeb na rozhraní (uzavřeného) systému se známým profilem použití:

- Garantovaná doba odezvy max. 10 ms v 99.9% případů.
- Spolehlivost (bezchybný provoz) 99.9% v rámci každého volání služby.

Pokud je část kontextu neznámá – typicky v případě jednotlivých komponent:

- Assume-Guarantee specifikace – předpoklad na kontext, garance kvality služby
- Parametrizovaná specifikace – neznámé informace nahrazeny parametry

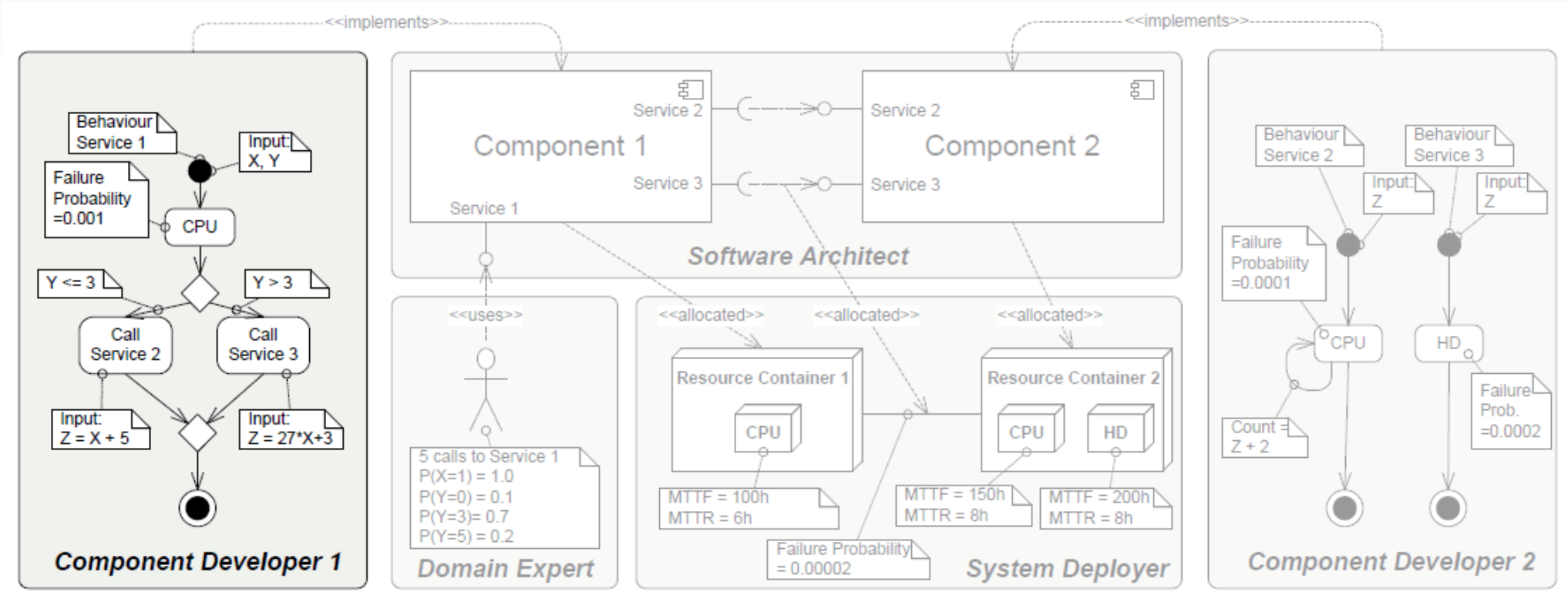
Kvality z pohledu ARCHITEKTA – neváží se na konkrétní službu, ale na zdroje systému, komponenty nebo systém jako celek

- Zatížení (resource utilization) konkrétního CPU nepřesáhne 90%.
- Zatížení CPU1 a CPU2 je rovnoměrně rozloženo.
- Konkrétní CPU je dostupné (available) 99.9% v rámci každého měsíce

Příklad – Assume-Guarantee specifikace

Specifikace spolehlivosti (Rel) `Service1()`:

- **Assumptions:**
  - Rel `CPU` ≥ 94.33%; Y ≤ 3 v 80% případů
  - Rel `Service2()` ≥ 93.77%; Rel `Service3()` ≥ 96.13%
- **Guarantee:**
  - Rel `Service1()` ≥ 88.81%



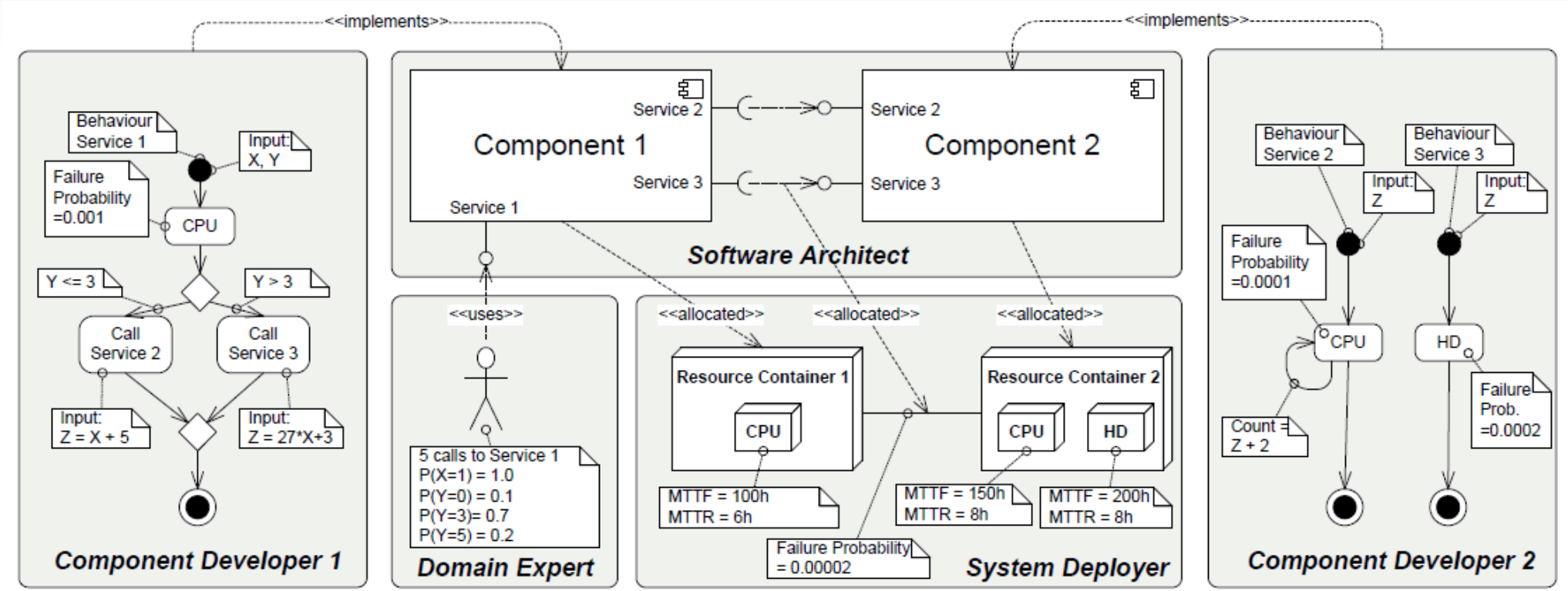
Příklad – Parametrizovaná specifikace

Problém Assume-Guarantee specifikace:

- Nerealistické předpokládat znalost všech assumptions
- Silná závislost na přesnosti assumptions → hrozí výrazné zkreslení

Parametrizovaná specifikace:

- Neznámé informace o kontextu nahrazeny parametry
- Výsledek – kompletní sada modelů jednotlivých součástí kontextu



Service Level Agreement (SLA)

The SLA records a common understanding about services, priorities, responsibilities, guarantees, and warranties.

The SLA may specify the levels of availability, serviceability, performance, operation, or other attributes of the service, such as billing.

Charakteristiky:

- Komplexnější než QoS – pohled na službu ve stylu SOA, kdy za využití služby platím, a proto chci jisté garance
- Často doprovázeno penalizacemi poskytovatele v případě nedodržení SLA
- Inspirováno call centry a service desk systémy
- Častější uplatnění v SOA než v CBSE

Shrnutí

- Definice softwarové komponenty
- Charakteristiky SW komponent
  - Společné s mechanickými komponentami
  - Odlišné od mechanických komponent
  - Komponenta vs. objekt/třída
- Middleware pro komponentové systémy
  - Komponentové frameworky
  - Komponentové modely
- Komponentový vývoj
  - Základní business pohled
  - Komplexní vývojářský pohled
  - Vývojářské modely a role
  - Dokumentace rozhraní komponent
    - Quality of Service (QoS)
    - Service Level Agreement (SLA)