|  |  |  |
| --- | --- | --- |
|  | **Софийски университет „Св. Кл. Охридски”**  Факултет по математика и информатика  *Магистърска програма „Софтуерни технологии”* |  |

**Предмет: Обектно-ориентиран анализ и проектиране   
на софтуерни системи**

***Зимен семестър, 2020/2021 год.***

**Тема 3: UML и ролята на SOA архитекта**

**Есе**

*Автор:*

*Десислава Свиленова Ташева*

*фак. номер* 26472

декември, 2020

София

Съдържание

[1 Въведение 2](#_Toc59476871)

[2 Характеристики на SOA 2](#_Toc59476872)

[2.1 Дефиниция 2](#_Toc59476873)

[2.2 Проблеми, които могат да настъпят при изграждането на SOA 3](#_Toc59476874)

[3 Model-Driven Architecture 3](#_Toc59476875)

[3.1 Описание на нивата 3](#_Toc59476876)

[3.2 Метамодели 4](#_Toc59476877)

[4 Agile методологии и SOA 5](#_Toc59476878)

[4.1 Основни Agile принципи: 5](#_Toc59476879)

[4.2 Agile SOA: 5](#_Toc59476880)

[5 „4+1“ модел 5](#_Toc59476881)

[5.1 Описание 5](#_Toc59476882)

[5.2 Описание на изгледите при „4+1“ 6](#_Toc59476883)

[5.3 Връзки между изгледите 7](#_Toc59476884)

[6 „1+5“ модел 7](#_Toc59476885)

[6.1 Описание 7](#_Toc59476886)

[7 Стереотипи 8](#_Toc59476887)

[7.1 Дефиниция за стереотипите и метакласовете 9](#_Toc59476888)

[7.2 Нотация 9](#_Toc59476889)

[8 Заключение и очаквано бъдещо развитие 10](#_Toc59476890)

[9 Използвани литературни източници 10](#_Toc59476891)

# Въведение

Есето има за цел да покаже как могат да се свържат компонентите на SOA архитектурния модел като се използва UML нотация, за да се създадат SOA сценарии.

# Характеристики на SOA

## Дефиниция

Ориентираната към услуги архитектура (SOA) е подход за изграждане на сложни софтуерни системи от група услуги, които могат да се преизползват и се подчиняват на SOA принципи. Някои от тях са :

* [**Стандартизирани условия на услугите**](https://bg.wikipedia.org/w/index.php?title=%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%B8%D0%B7%D0%B8%D1%80%D0%B0%D0%BD%D0%B8_%D1%83%D1%81%D0%BB%D0%BE%D0%B2%D0%B8%D1%8F_%D0%BD%D0%B0_%D1%83%D1%81%D0%BB%D1%83%D0%B3%D0%B8%D1%82%D0%B5&action=edit&redlink=1)
* **Автономност на връзката между услугите**
* **Дълголетие на услугата**
* [**Абстрактни услуги**](https://bg.wikipedia.org/w/index.php?title=%D0%90%D0%B1%D1%81%D1%82%D1%80%D0%B0%D0%BA%D1%82%D0%BD%D0%B8_%D1%83%D1%81%D0%BB%D1%83%D0%B3%D0%B8&action=edit&redlink=1)
* [**Автономни услуги**](https://bg.wikipedia.org/w/index.php?title=%D0%90%D0%B2%D1%82%D0%BE%D0%BD%D0%BE%D0%BC%D0%BD%D0%B8_%D1%83%D1%81%D0%BB%D1%83%D0%B3%D0%B8&action=edit&redlink=1)
* **Едно извикване на услугата не трябва да влияе на друго такова**
* **Нормализиране на услугата**
* **Композитност на услугите**
* [**Енкапсулация**](https://bg.wikipedia.org/w/index.php?title=%D0%95%D0%BD%D0%BA%D0%B0%D0%BF%D1%81%D1%83%D0%BB%D0%B0%D1%86%D0%B8%D1%8F&action=edit&redlink=1)**на услугитe**

Смята се, че SOA превръща бизнеса в „по-гъвкав“, с други думи ни позволява по-бързо да реагираме на промени в условията, което няма да доведе до много загуби на ресурси и средства. Позволява ни да конструираме приложения от преизползваема функционалност, която може да бъде създадена бързо, главно от вече съществуващи услуги. По този начин построяването на услуга се отделя от разработването на приложението, което я използва. Различните компоненти си комуникират чрез портове за комуникация в мрежата и всеки един от тях изпълнява само едно действие. Понеже отделните компоненти се разработват без да се знае кой точно ще ги ползва, се избягва зависимости от операционната система или технологията с която са разработени. Използват се множество стандари като например XML (extensible markup language) and SOAP (simple object access protocol). По този начин модифицирането на интерфейсите на една от услугите, не засяга друга услуга.

## Проблеми, които могат да настъпят при изграждането на SOA

Когато се проектират интеграционни решения, е от съществено значение архитекта да бъде в състояние да моделира пълното архитектурно описание. За тази цел са нужни:

- модел на архитектурните изгледи, който позволява моделирането на интеграционни платформи - набор от елементи на модела, които представят презентирането на цялата архитектура на платформата.

Използването на много различни услуги, прави описанието сложно, затова е нужно да се използват добре дефинирани компоненти за моделиране, за да може да се опишат моделите и връзките между тях.

# Model-Driven Architecture

За да се намали сложността при използването на SOA, е нужно да се следва методологичнен подход. Най-разпространеният от тях, който се ползва сред тях е ориентираният към моделите. Идеята, която стои зад него е, че предоставя насоки и структурира спецификациите, които са представени чрез модели. MDA разделя процеса на разработка на три абстрактни слоя – CIM (Computation Independent Models), PIM (Platform Independent Models), PSM (Platform Specific Models). Най-важната част е връзката между моделите дефинирани на всяко ниво и тези между различните нива, което позволява автоматизация на процеса на разработка.

## Описание на нивата

1. **Computation-independent models (CIM)** - това са изгледи на системата от независима гледна точка на изчисленията. Тя се фокусира върху средата на системата и изискванията към нея. Детайлите за структурата и процесите на системата са скрити. Понякога CIM се нарича домейн модел и се използва като речник за спецификацията на домейна.
2. **Platform-independent models (PIM)** – изглед на системата от независима гледна точка от платформата. Фокусира се върху работата на системата, като крие детайли за съответната платформа. Показва спецификации за системата, които не се променят в зависимост от платформата, която се използва. Постига се определена независимост от платформата, така че да се използва от множество други платформи от подобен тип.
3. **Platform-specific models (PSM)** – изглед на системата гледна точка, специфична за платформата. Комбинира PIM с допълнителен фокус върху използването на специфична платформа от системата.

MDA процеса започва с дефинирането на CIM или домейн модела, след това тези модели могат да се трансформират в PIM и PSM модели. Тези трансформации се наричат вертикални трансформации. Допълнителни трансформации се използват за преобразуването на по-абстрактен PIM модел, в по-детайлен такъв. Когато се прилага в SOA, CIM-нивото може да опише бизнес моделите, които включват целите, бизнес правилата, процесите и услугите. Това може да се направи с например Business Motivation Model (BMM) или Business Process Modeling Notation (BPMN). PIM нивото описва SOA моделите, които са независими от платформата/технологията за изпълнение, която се използва. Включва модели на интерфейсите на услугите, контрактите на услугите, участниците и др. Моделите от това ниво се считат за модели на софтуерната спецификация. PSM нивото описва SOA платформените модели като изпълними артефакти. Тези модели могат да бъдат смятани за модели за реализация на софтуера. Ако SOA е имплементирано с например уеб услуги, интерфейсите на тези услуги трябва да могат да бъдат представени като се използва Web Service Definition Language (WSDL).

## Метамодели

Метамоделите за MDE (Model-driven engineering) са важна концепция, която може да се приеме за описание как се построяват моделите. За да се приложи MDE (MDA в частност) трябва да се моделира метамодел за SOA приложението. За да се случи това, първо е нужен механизъм за дефиниране на език за моделите, който да помогне на инструмента за трансформация да прочете, напише и разбере моделите. След това се определят правила за трансформация, които посочват как модел на изходния език, може да се трансформира в модел на целевия език. Тези правила използват метамодела на изходния и на целевия език, за да дефинират трансформацията. Теза метамодели наприме може да се използват за трансформиране на PIM модели към PSM модели, генериране на специфични модели за определени изгледи за различните заинтересоване лица и др.

# Agile методологии и SOA

## Основни Agile принципи:

* **Хората и взаимодействията между тях са преди инструментите, които те използват**
* **Работещият софтуер е с приоритет пред документацията**
* **Фокусиране върху колаборацията с клиента**
* **Трябва да може бързо да се отговори на някаква промяна**

## Agile SOA:

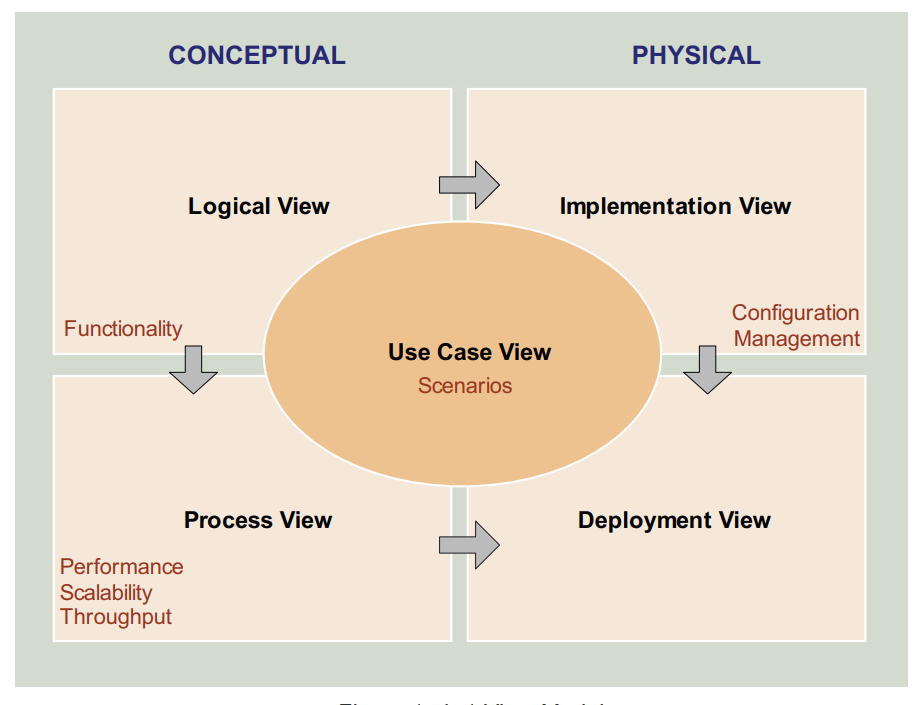
Някои от Agile принципите са приложими и за SAO, като на места могат да бъдат променени, за да могат да се впишат в принципите на SOA.

* **Architecture-driven iterative approach —** този подход се справя с постоянно променящите се условия и бизнес изисквания, които също така могат да бъдат неясни.
* **Governance-driven reuse –** фокусът е върху това да се позволи на софтуера да посреща променящи се условия и изисквания и да се адаптира към тях. Управлението на ресурсите тук също е важно, целта е бизнеса да получи максимална печалба от това, което се произвежда.
* **Metadata-driven development —** Документацията и метадата на услугите са свързани, понеже и двете представляват артефакти, които изпълняват важна роля при жизнения цикъл на услугата. Има една съществена разлика – софтуерът се използва от хората, а метадатата се използва от машините. Ако документацията е във вида на метадата, тя следва Agile принципите.

# „4+1“ модел

## Описание

Архитектурата означава различни неща за различните заинтересовани страни. Например системният инженер би се интересувал единствено от хардуера и конфигурацията на системата, ръководителят на проекта от ключовите компоненти, които трябва да се разработят и сроковете, в които това трябва да се случи, за програмистите са важни класовете, които образуват компонента, а тестерът се интересува от сценариите, които са възможни при използването на системата. Именно затова архитектът трябва да представи различни изгледи на архитектурата за различните заинтересовани страни, като показва важните неща, докато маскира нещата, които не са нужни. „4+1“ е архитектурен подход, който организира архитектурата на приложението в различни изгледи, за да задоволят нуждите на конкретните заинтересовани лица.



## Описание на изгледите при „4+1“

* **Logical view**

Този изглед се фокусира върху реализацията на функционалност по отношение на структурни елементи, ключови абстракции и механизми, разделяне на проблемите и разпределение на отговорностите. Архитектите използват този изглед за изследване на функционалността, от която се нуждаят крайните потребители. В една обектно-ориентирана система това е на ниво клас. В по-комплексните системи може да има нужда от изглед на пакетите, след което пакетите се декомпозират в клас диаграми.

* **Implementation view (Development view)**

Този изглед се фокусира върху зависимостите и връзките между компонентите, как са организирани модулите и как се преизползват. Сравнено с логическия изглед, който се намира на концептуално ниво, диаграмите в това ниво репрезентират артефактите на физическото ниво, които са имплементирани от екипа. Използват се **диаграми на компонентите**.

* **Process view**

Този изглед представя нефункционалните изисквания за проекта като производителност, мащабируемост и др. Всеки процес в системата се състои от задачи, които се изпълняват в колаборация с други структурни елементи (От логическия изглед). Архитектурата на процесите може да се покаже с много нива на абстракция като например взаимодействието между системите, подсистемите и обектите и т.н. Всичко това зависи от нуждата на заинтересованите лица. Най-често се използват **диаграмите на дейностите, диаграма на последователността и диаграма на комуникациите**, за да се опише този изглед.

* **Deployment view**

Този изглед е за системните администратори, които трябва да разберат физическата локация на софтуера, физическите връзки между възлите, внедряването и инсталирането. Всички елементи като процеси, задачи и обекти трябва да се свържат с възлите, които ще ги изпълнят. Този изглед предоставя всички възможни хардуерни конфигурации и свързва компонентите от изгледа за имплементацията с тях. Използват се **диаграми за внедряване** (Deployment diagram)

* **Use Case view**

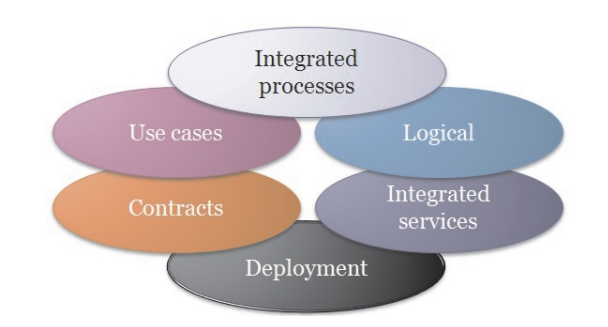
Този изглед показва потребителските случаи, които описват поведението на системата през очите на крайните потребители и други заинтересовани лица. Този изглед трябва да е първи при създаването на всички изгледи, защото описва сценариите, които обединяват останалите изгледи заедно. Използва се Use Case диаграма, също така диаграма на дейностите.

## Връзки между изгледите

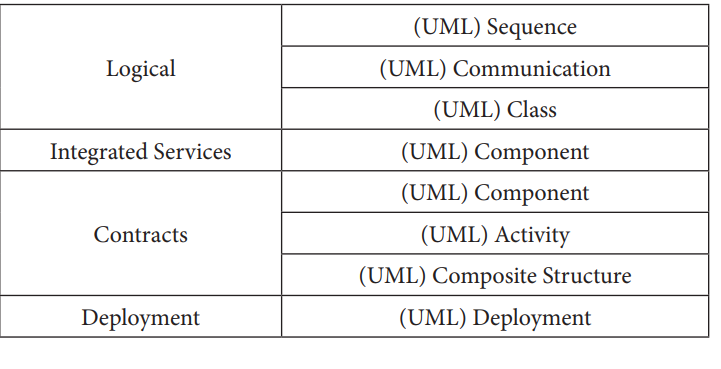
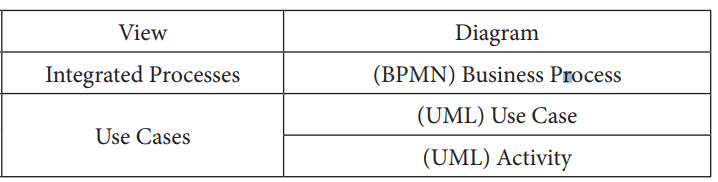
Логическият изглед и изгледът на процесите се намират но концептуално ниво и се използват, за да се направи дизайн на системата. Изгледът на имплементацията и на внедряването са на физическо ниво и представляват истинските компоненти на системата и как те ще бъдат внедрени. Логическият изглед и изгледът на имплементацията са логически свързани и са близки до функционалност. Те определят как ще бъде моделирана и имплементирана функционалността. Сценариите се следват при анализирането на логическия изглед, когато се имплементират компонентите и се реализират в изгледа на процесите, след което се внедряват чрез изгледа за внедряване на системата.

# „1+5“ модел

## Описание



Основният изглед на този модел е изгледът за интегрираните процеси. В този изглед се моделират бизнес процеси, които са посветени на автоматизацията на интеграционната платформа. Другите четири изгледа (Use Cases, Logical, Integrated Services и Contracts) представят дизайна на платформата за интеграция. Изгледът за потребителските случаи съдържа функционалните изисквания за системата, които се интегрират в платформата. Изгледът с интегрираните услуги представя услугите, които се предоставят от информационната система и как те са свързани към шината. Contracts изгледа предоставя информация за компонентите на системата и какви ограничения са дефинирани между тях. Изгледът за внедряване е същия като при другия модел и показва как системата се интегрира на различни среди за изпълнение.



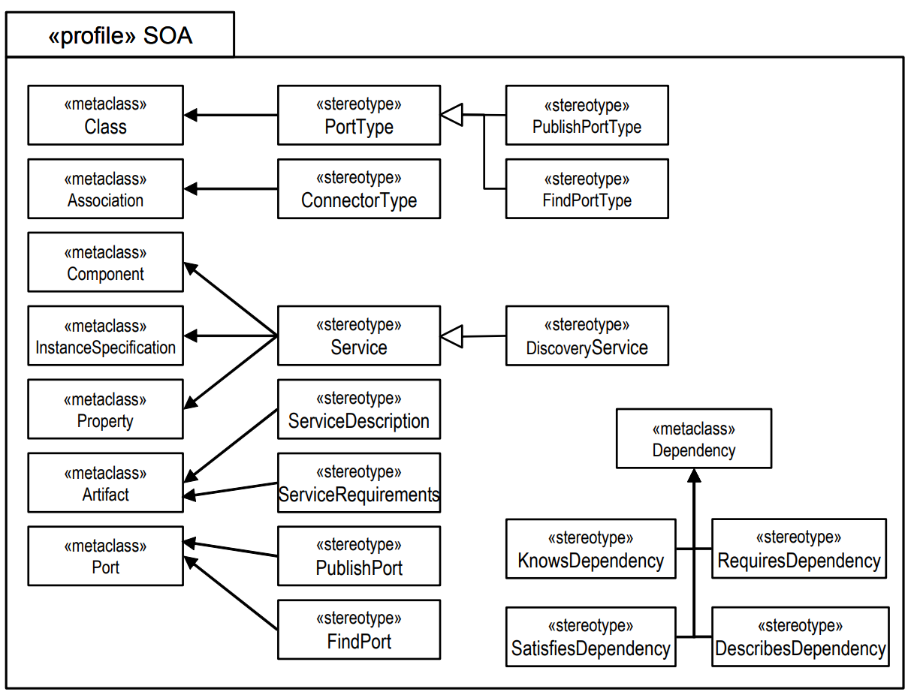
# Стереотипи

UML се използва за дизайн, спецификация и документация на атрефакти, които се създават по време на процеса по разработването на информационна система. Освен това разполага с механизми, които поддържат разрастването на нейната семантика. Най-базовата структура, която позволява подобно разширение е стереотипът. Колекцията от стереотипи пък може да образува профил. Например SoaML е пример за UML профил, който спомага за описанието на архитектура ориентирана към услугите.

## Дефиниция за стереотипите и метакласовете

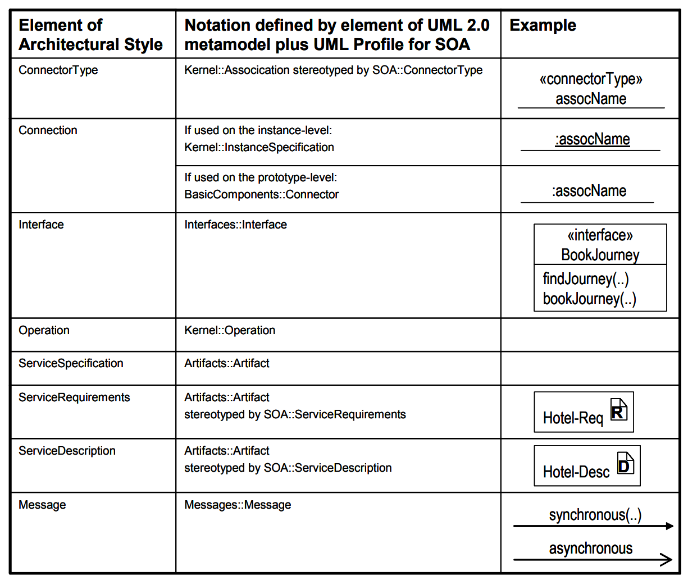
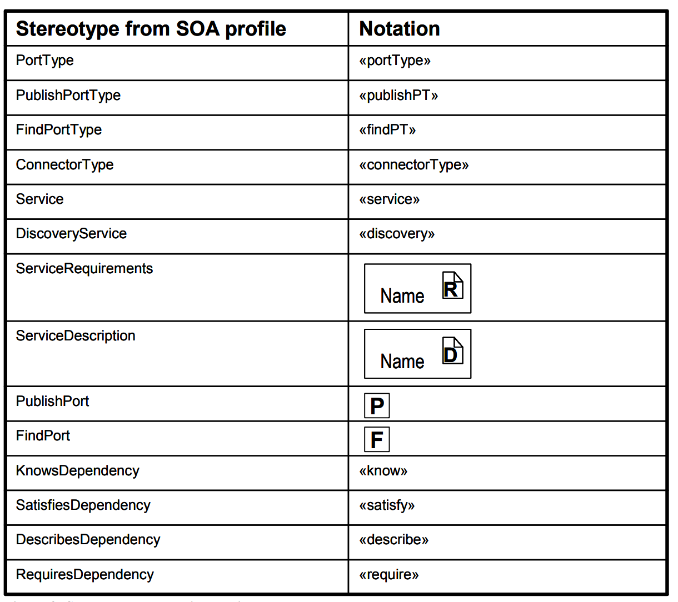
Метакласовете e клас на профила, който може да бъде разширен от стереотипи. Това се случва чрез специална връзка наречена „разширение“ и позволява лесно да се разширяват класове и ако се наложи в бъдеще, тези връзки могат да бъдат лесно премахвани в бъдеще.

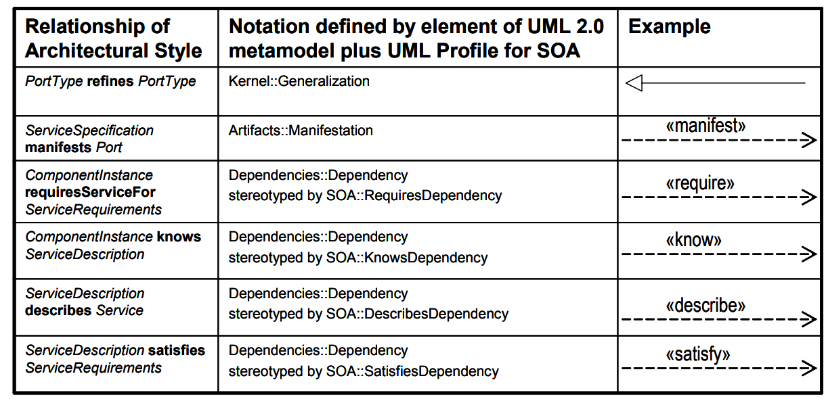
Пример за връзки между метакласове и стереотипи в една SOA архитектура:



## Нотация

Таблица с част от нотацията на стереотипи, свързани със SOA архитектурата





# Заключение и очаквано бъдещо развитие

# Използвани литературни източници

1. <https://www.utwente.nl/en/eemcs/trese/graduation_projects/2012/RT-002.pdf>
2. <https://www.uml-diagrams.org>
3. <https://lists.oasis-open.org/archives/soa-rm/200610/pdf00000.pdf>
4. <https://www.omg.org/spec/SoaML/1.0/Beta2/PDF>
5. <https://en.wikipedia.org/wiki/Service-oriented_architecture>