UNIFIED MODELING LANGUAGE (UML)

SOFTWARE ENGINEERING 2

Developed by :Malihe Hashemi
Supervisor : Ahmad abdollahzade

Intelligent systems lab (http://ce.aut.ac.ir/islab)

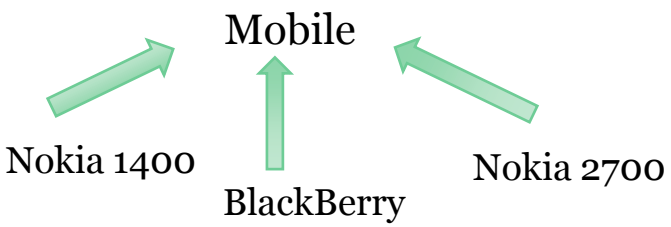
OBJECT ORIENTED BASIC PRINCIPLES

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Abstraction** (انتزاع):
  - برجسته کردن اطلاعاتي از يك موجوديت در رابطه با يك كاربرد خاص و حذف اطلاعاتي غيرضروری. **(تمرکز بر ویژگی‌های کلیدی و ضروری، بدون درگیر شدن در جزئیات غیرضروری و بدون اهمیت)**
  - از بین بردن وجوه تمایز و تاکید بر مشترکات.
  - انتزاع یکی از مهمترین راه های مقابله با پیچیدگی است (زیرا تنها بر روی ویژگی‌ها موردنظر و مرتبط با مسئله تمرکز می‌شود).
  - Object oriented به روش‌های گوناگون از انتزاع پشتیبانی می کند:
    - **generalization/specialization**
    - **aggregation/decomposition**

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Abstraction** (انتزاع):
- **Generalization/specialization**: بیان کننده ی رابطه Is-A است که در آن یک عنصر نوعی از عنصر دیگر است.
  - در Generalization ویژگی های مشترک بین موجودیت های مرتبط با یکدیگر استخراج می‌شود و در قالب یک موجودیت سطح بالاتر دسته بندی می شود.
  - در specialization بر خلاف Generalization از روی یک موجودیت سطح بالاتر یک instance به خصوص ایجاد می‌شود که علاوه بر ویژگی‌های این موجودیت شامل ویژگی‌های جزئی تری نیز می باشد.




```
public class MobilePhone {
    public void Calling(){}
    public void SendSMS(){}
}
```

```
public class Nokia1400 extends MobilePhone{

}
```

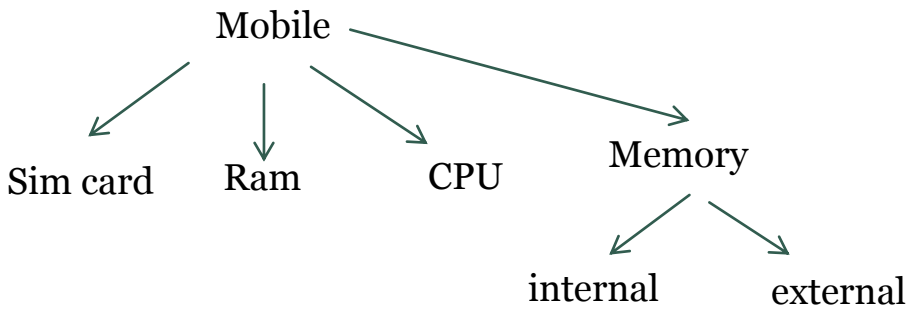
```
public class Nokia2700 extends MobilePhone {
    public void FMRadio(){}
    public void MP3(){}
    public void Camera(){}
}
```

```
public class BlackBerry extends MobilePhone{
    public void FMRadio(){}
    public void MP3(){}
    public void Camera(){}
    public void Recording(){}
    public void ReadAndSendEmails(){}

}
```

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Abstraction** (انتزاع):
- **aggregation/decomposition**: بیان کننده‌ی رابطه part-whole و یا Has-A است.
  - Aggreagtion: در آن عناصر کوچکتر در کنار هم یک عنصر بزرگتر را می‌سازد.
  - Decomposition: برخلاف aggregation ، در decomposition عنصر سطح بالاتر به عناصر کوچک تر تجزیه می شود.



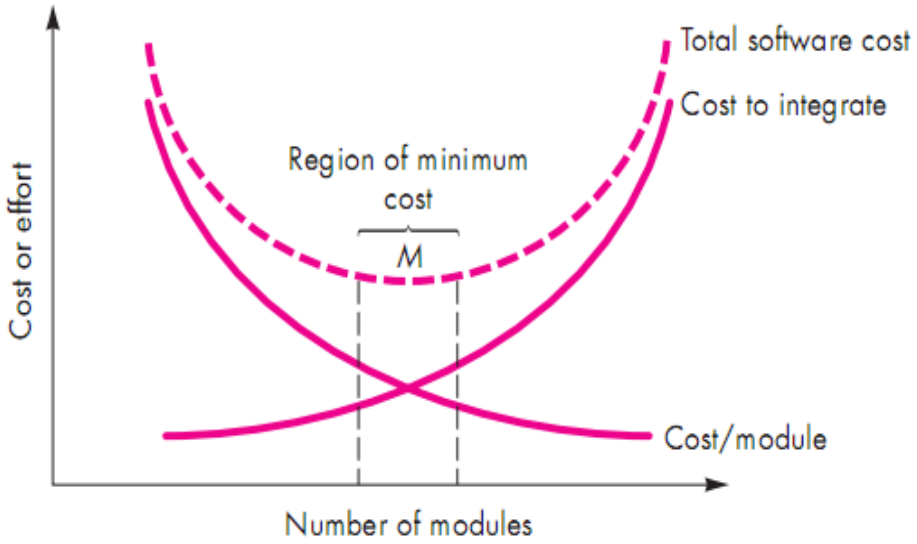
OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Encapsulation (محصورسازي):**
  - به آن information hiding نیز گفته می شود.
  - در برنامه نویسی شی گرا، object ها با یکدیگر از طریق message passing در ارتباط هستند. در نتیجه یک object تنها interface یک object دیگر را می‌شناسد و از این طریق operation های آن را فراخوانی می‌کند. تنها operation های یک obeject می‌توانند به داده های آن دسترسی داشته باشند و وضعیت آن را تغییر دهند. بنابراین وضعیت داخلی و منطق موجود در operation های object به طور مستقیم قابل دسترسی نبوده و خارج از این object، visible نیستند. (به عبارت دیگر یک object از طریق interface ، داده و منطق عملیاتی خود را پنهان می‌کند.)
  - **فواید:**
    - جداسازي رفتار از نحوه‌ی پیاده سازی آن
    - کاهش تأثیر تغییرات یک object بر روي استفاده کننده‌های آن
    - استانداردسازي
    - و ...

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Modularity:**
  - تقسيم بندي يك المان پيچيده به مجموعه‌اي از المان‌هاي ساده بقسمي كه قابل مديريت و كنترل باشند.
  - ماژولاریتی یکی از مهمترین اصولی است که separation of concern را محقق می‌سازد.
  - **فواید**
    - كاهش درجه پيچيدگي
    - افزایش مدیریت و کنترل (برنامه ریزی و work assignment به نحو بهتری انجام می پذیرد.)
    - تست و debugging به نحو بهتری انجام می پذیرد.
    - اعمال تغییرات با هزینه و زمان کمتر (در صورتی که ماژولاریتی به نحو موثر انجام پذیرفته باشد، side effrct ها کاهش می بیابد.)
    - افزایش قابلیت استفاده مجدد
- **تمرکز بر (functional independence):**
  - **High Cohesion**
    - افزایش Reusability
  - **Low Coupling**
    - كاهش تأثيرات تغييرات دروني يك المان بر ساير المان‌ها

OBJECT ORIENTED BASIC PRINCIPLES(CONT)



- آیا در صورتی که سیستم را تا حد ممکن بدون در نظر گرفتن هیچ محدودیتی ماژول بندی کنیم، effort تولید سیستم به تناسب آن کاهش می‌یابد؟
- (شکل مقابل نشان می دهد که این مطلب صادق نیست)

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Dynamic Binding**
  - **Binding به معنای مرتبط کردن method call با یک method body مشخص است.**
  - **Binding به دو صورت انجام می پذیرد:**
    - Static: در آن binding قبل از اجرای برنامه و در زمان کامپایل انجام می‌پذیرد.
    - Dynamic: در آن binding در run time رخ می‌دهد.

```
public class A {

    public void doIt(){}

}

public class B extends A{

    public class doIt()

}
```

```
public class Test {
    public static void main(String[] args) {
        A x = new B();
        x. doIt();
    }
}
```

کامپایلر نمی‌تواند type عنصر x را تعیین کند. (یک object از type B ایجاد شده و reference آن به یک عنصر از type A مرتبط شده است، بنابراین یک error ایجاد می‌شود. اما کامپایلر این error را در نظر نمی‌گیرد و تعیین type عنصر x را به run time واگذار می‌نماید.
در این حالت در زمان اجرا متد doIt اجرا شده مربوط به کلاس B است.

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

```
public class BankingAccount {
    public void debit(int amount){}
}

public class SavingAccount extends BankingAccount{
    public void debit(int amount){}
}

public class CheckingAccount extends BankingAccount{
    public void debit(int amount){}
}

public class TestClass {
    public void debitAccount (BankingAccount a, int amount){
        a.debit(amount);
    }
    public static void main(String[] args) {
        TestClass t = new TestClass();
        /* ... get account type  a and money from user */
        t.debitAccount(a, amount);
    }
}
```

- **Dynamic Binding**
  - مثال دیگر

OBJECT ORIENTED BASIC PRINCIPLES(CONT)

- **Polymorphism (چند ریختی):**
  - ارسال يك پيغام واحد به المان‌هاي مختلف و دريافت چندريخت جواب
  - Over Riding:
    - چند تابع با اسم و كاركرد يكسان اما روش‌هاي متفاوت براي انجام عملكرد
  - Over Loading:
    - چند تابع با اسم يكسان و پارامترهای ورودی متفاوت و كاركردهاي متفاوت و متناسب با ورودی
  - **فواید:**
    - افزايش انعطاف‌پذيري سيستم در مقابل توسعه و تعميم (توسعه سيستم بدون ايجاد تغيير)
    - جابه جايی object ها با interface يكسان در زمان اجرا
    - ايجاد Framework

Overloading

```
import java.math.BigDecimal;


public class DoublePrinting {

    public void print(Double r){
        System.out.println(r);
    }
    public void print(Double r , int precision){
        Double truncated_r = new BigDecimal(r).setScale(
                precision,BigDecimal.ROUND_HALF_UP).doubleValue();
        System.out.println(truncated_r);
    }

    public static void main(String[] args) {

        DoublePrinting doublePrint = new DoublePrinting();
        doublePrint.print(123.55567);
        doublePrint.print(123.55567,3);
    }
}
```

```
123.55567
123.556
```

OBJECT ORIENTED PROGRAMMING VS. STRUCTURA L PROGRAMMING

- روش شيءگرا
  - سيستم به صورت مجموعه‌اي از Objectها تعريف می شود.
  - درصدد نزديك شدن به دنياي واقعي
  - مكانيزم ارتباطي
    - Message Passing
  - Class: ساختار و رفتار مشترك اشياء مشابه در كلاس متناظر آن‌ها تعريف مي‌شود
- روش رويه‌اي (Procedural)
  - فراخواني .Proc‌ها و .Func‌ها
  - جدايي Data Structure از رفتارها

MODELING

- مدل ساده شده یک واقعیت است.
- مدل دارای abstraction است. مدل جنبه‌های مهم را از یک view به خصوص مورد توجه قرار می‌دهد و سایر جزئیات غیرضروری را ساده کرده و یا به آن ها نمی‌پردازد.
- مدل وسیله بازنمایی است:
  - Presentation (syntax) +Semantic
- **چرا باید مدل سازی کرد؟**
  - ممکن است دامنه مسئله به خوبی شناخته نشده باشد.
  - نیازمندی ها به درستی مشخص نشده باشند.
  - راه حل دارای پیچیدگی‌هایی باشد. (با استفاده از مدل می توان راه حل های مختلف را بررسی کرد، گزینه های مختلف را بررسی نمود و از طریق تکامل گزینه موردنظر به راه حل مناسب دست یافت.)
  - نیاز به نگه‌داری سیستم وجود داشته باشد.
  - scope نیازمندی‌های سیستم محدود نبوده و دچار تغییرات گردد.
  - افراد زیادی در پروژه درگیر باشند و هماهنگی و تعامل بین آن‌ها از اهمیت ویژه‌ای برخوردار باشد.
  - سیستم دارای حساسیت زیادی باشد، و باید پیش از ساخت از درست بودن راه حل موردنظر اطمینان حاصل کرد.
- **چه تعداد مدل؟**
  - بستگي به پيچيدگي و ابعاد سيستم دارد.

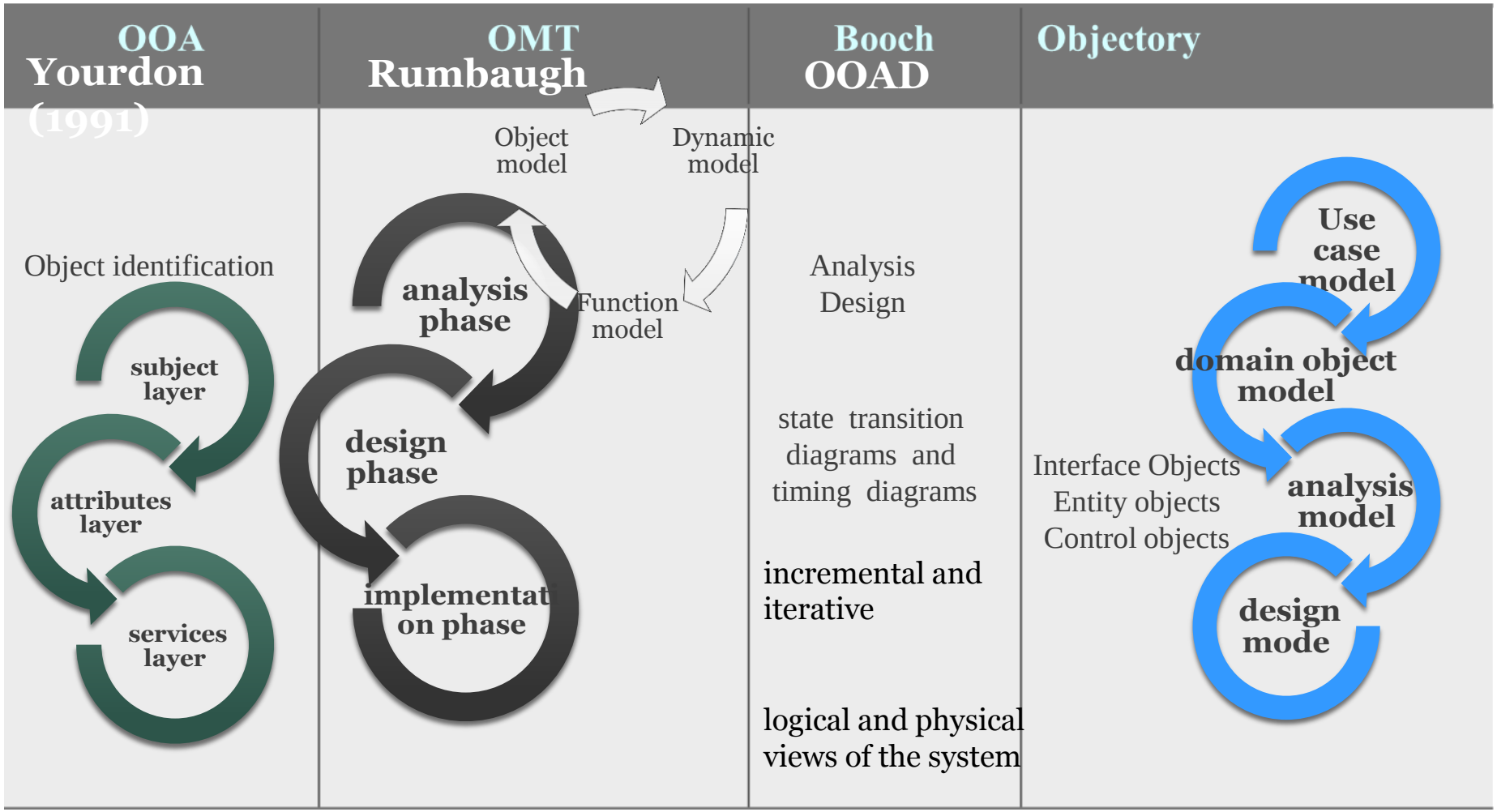
MODELING

- اصول مدل سازی
  - مدل ها در سطوح انتزاع مختلف، با جزئیات و دقت متفاوت تولید می‌شوند.
  - بهترین مدل شبیه ترین مدل به واقعیت است. (در عین حال که **abstract** است.)
  - برای تحلیل و طراحی سیستم یک مدل کافی نیست. برای مدل کردن از **چند دید (View)** به سیستم نگاه مي‌کنیم.
  - برخی از اصول مدل سازی در **agile**
    - هدف اصلی تیم نرم افزار تولید نرم افزار است نه تولید مدل.
    - بیش تر از حد نیاز مدل نکنید.
    - ساده ترین مدل را برای توصیف مساله و نرم افزار موردنظر ایجاد نمایید.
    - مدل ها باید طوری ایجاد شوند که نسبت به تغییرات پاسخگو باشند.

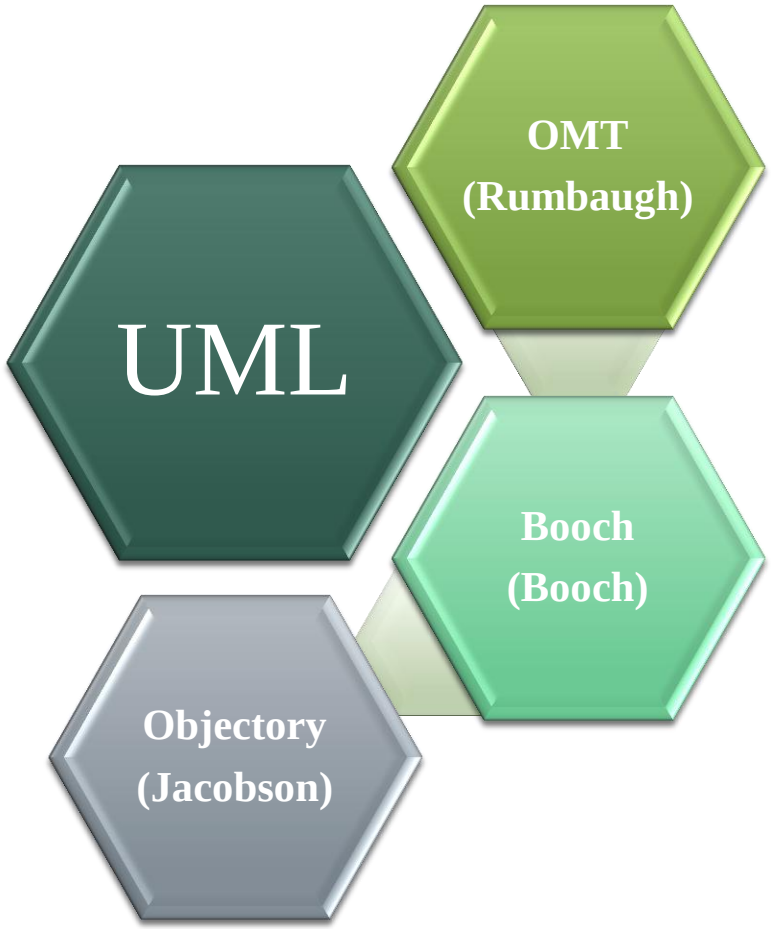
MODELING

- **فواید مدل سازی**
  - **کنترل پیچیدگی**
    - مدل را می‌توان در سطح abstraction مناسب رسم نمود و وارد جزئیات غیرضرروی نشد.
    - با استفاده از مدل می‌توان تغییرات را مدیریت کرد و با بررسی تاثیر تغییر در سایر بخش‌های سیستم هزینه این تغییر را بررسی نمود.
    - می توان سیستم را طوری طراحی نمود که هزینه پاسخ گویی به تغییرات کم شود.
  - **بررسی هرینه و ریسک گزینه‌های مختلف طراحی و انتخاب گزینه بهتر:** بدون مدل‌سازی سیستم در آغاز نمی‌توان گزینه‌های طراحی مختلف را در نظر گرفت و آن‌ها را نسبت به یکدیگر مقایسه کرد.
  - **مدل‌سازی می‌تواند بهره‌وری تیم تولید را افزایش دهد:** می‌توان مدل‌های تهیه شده را با استفاده از روش‌های model transformation و model execution به کد تبدیل نمود.
  - **افزایش کیفیت نرم افزار:** با استفاده از مدل‌سازی می‌توان defect های موجود در محصول را کاهش داد. با توجه به اینکه مدل‌ها پیش از تولید محصول مرور و بازبینی می‌شوند، خطاهای موجود سریع تر شناسایی می‌شود.
  - **افزایش قابلیت استفاده مجدد**: در صورتی‌که سیستم مدل شده باشد، می‌توان در تهیه سیستم‌های مشابه، مدل‌های طراحی شده را مورد استفاده مجدد قرار داد.
  - **افزایش تعامل و همکاری بین اعضای تیم**: مدل‌های تهیه شده یک زبان مشترک برای برقراری ارتباط بین اعضای مختلف تیم تولید فراهم می‌آورد.
  - **مستندسازی**

UML HISTORY


OOA
Yourdon
(1991)
Object identification
subject layer
attributes layer
services layer
OMT
Rumbaugh
Object model
Dynamic model
Function model
analysis phase
design phase
implementation phase
Booch
OOAD
Analysis
Design
state transition diagrams and timing diagrams
incremental and iterative
logical and physical views of the system
Objectory
Use case model
domain object model
Interface Objects
Entity objects
Control objects
analysis model
design mode

UML HISTORY



1997– UML 1.0✓
1999– UML 1.3✓
2003– UML 2.0✓

UML DIAGRAMS

3- Diagrams

1- Class Diagram

2- Object Diagram

3- Use case Diagram

4- Sequence Diagram

5- Collaboration Diagram

6- State Diagram

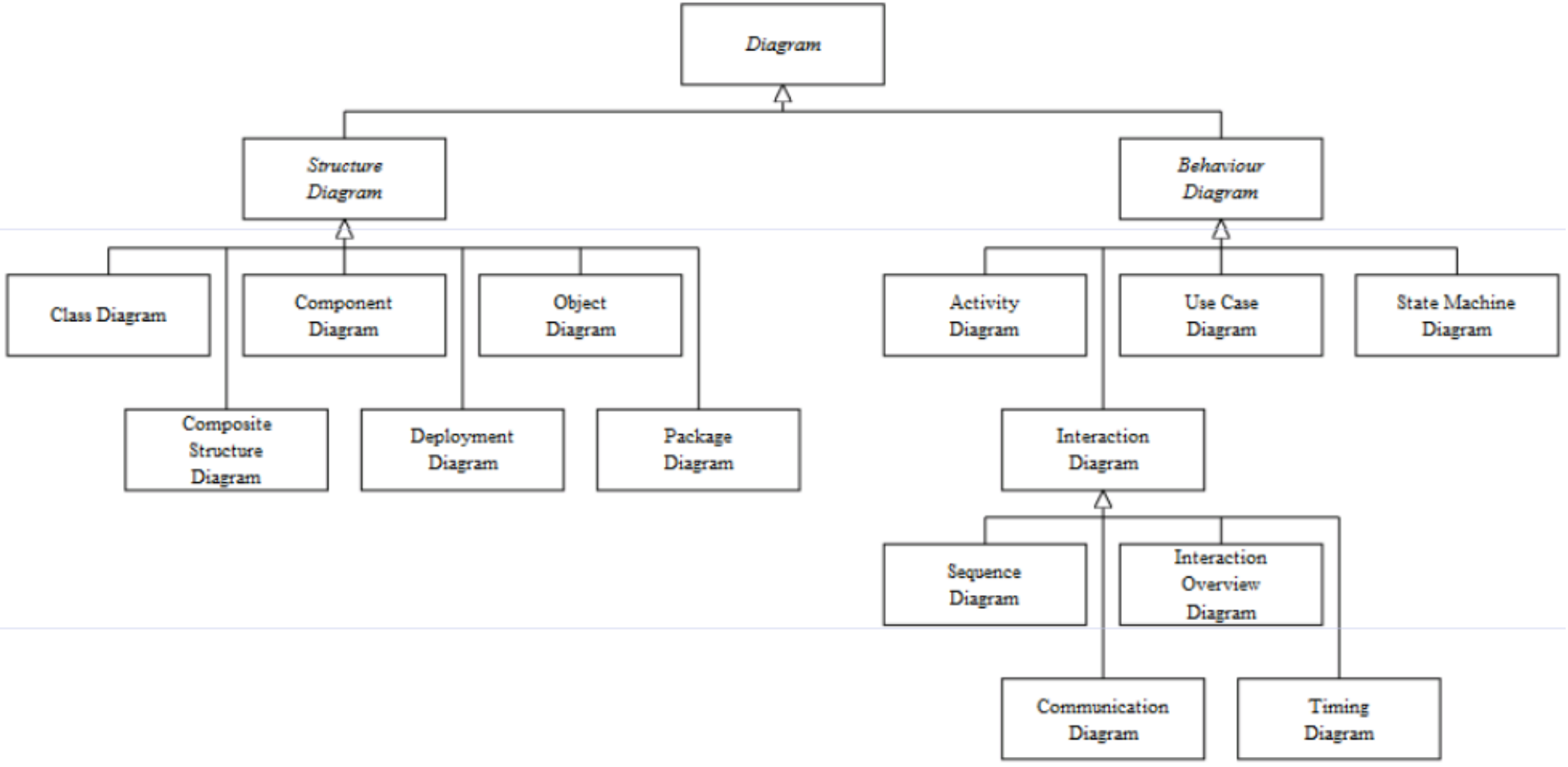
7- Activity Diagram

8- Component Diagram

9- Deployment Diagram

…

UML DIAGRAMS



UML BULIDING BLOCKS

Things (المان‌هاي مدل‌سازي)

→ Structural Things:

- Class
- Interface
- Use Case
- Component
- Node

→ Behavioral Things:

- Message
- State

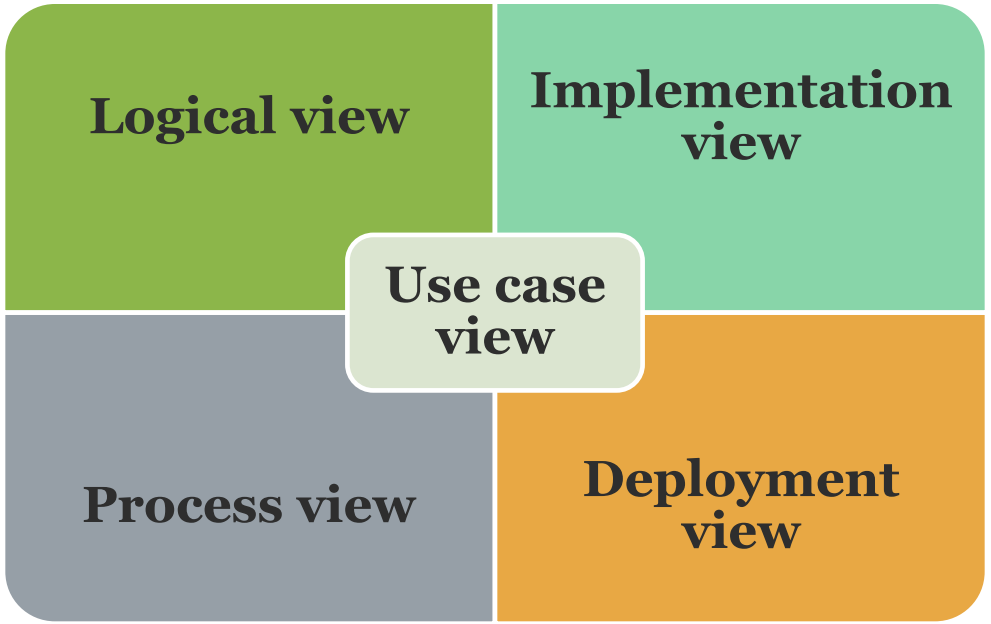
→ Grouping Things:

- Package

→ Annotational Things:

- Note

4 + 1 VIEW


Logical view
Implementation view
Use case view
Process view
Deployment view

4 + 1 VIEW

1- Use Case View:

- Requirements and Features (Functionality) - Analyst
- Use case & Activity & … diagrams

2- Logical View:

- Problem and Solution - Designer and Programmer
- Class & Sequence & Collaboration & State diagrams

3- Process (Concurrency) View:

- Multithreading (نمایش تعامل threadهایي که همزمان در سیستم زنده‌اند) - Programmer
- Activity Diagram

4- Implementation View:

- Technologies - Programmer
- Component diagram

5- Deployment View:

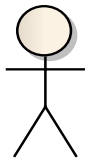
- Hardware and Topology - Designer and Technologist
- Deployment diagram

USE CASE DIAGRAM

- یک use case رفتار سیستم یا بخشی از سیستم را از دید موجودیت هایخارجی سیستم نشان می‌دهد. (به عبارتی یک use caseیک نتیجه observable برای موجودیت‌های خارجی سیستم فراهم می‌کند.)
- Use case با تعیین گام‌های موردنیاز برای دستیابی به یک هدف معین، مشخص می‌کند که کاربران به چه نحوی با سیستم در تعامل هستند. حالات استثنا ممکن است موجب ایجاد سناریوهایی مختلفی در این تعامل شود.
- - تعیین "چه" و عدم توجه به "چگونگي و نحوه‌ی پیاده‌سازی" در use case ها What not How
- نیازمندی‌های عملکردی سیستم را نشان می‌دهد و در طی فرآیند جمع‌آوری نیازهای سیستم برای تشخیص اینکه سیستم چه قابلیتهایی را دارد استفاده می‌شود.
- همچنین Use case در تولید test case نیز مورد استفاده قرار می‌گیرند.

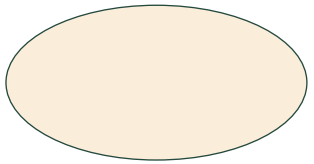
USE CASE DIAGRAM

- Actor
  - بیان کننده‌ی یک نقش است که توسط کاربران، سیستم و یا زیرسیستم‌هایی که خارج از سیستم قرار دارند ایفا می‌شود.
  - باید دقت کرد که actor یک کلاس به خصوص از نقش‌ها را تعریف می‌کند. هر نقش ممکن است توسط یک یا چند کاربر انجام شود و هر کاربر ممکن است در تعامل با سیستم دارای چندین نقش باشد.
  - هر actor با یک یا چند usecase در ارتباط است.



نحوه‌ی نمایش actor

- Use case: مجموعه عملياتي که توسط Actor انجام مي‌شود تا نتيجه‌اي معيني را توليد کرده يا در اختيار Actor قرار دهد.



نحوه‌ی نمایش use case

USE CASE DAIGRAM

- بصورت كلي براي رسم نمودارهاي مورد كاربرد سيستم بايد سه مورد را انجام داد:
  - شناسايي Actorها و ارتباط آن‌ها
  - شناسايي Use Caseها و ارتباط آن‌ها
  - تعيين ارتباط Actorها و Use Caseها

USE CASE DIAGRAM

- توجه به سوالات زیر به شناسایی actor های سیستم کمک می‌کند:
  - چه کسي در نيازمنديهاي مشخص سيستم ذينفع است ؟
  - سیستم در کدام بخش سازمان استفاده خواهد شد ؟
  - چه کسي تامين کننده اطلاعات سيستم است يا از اطلاعات سيستم استفاده ميکند و يا اطلاعات سيستم را حذف ميکند ؟
  - چه کسي از عملکردهاي مشخص سيستم استفاده ميکند ؟
  - چه کسي پشتيباني و نگهداري سيستم را به عهده خواهد داشت ؟
  - آيا سيستم از منبع يا وسيله خارجي استفاده ميکند ؟
  - آيا يک موجوديت (کاربر ، سيستم ، فرآيند و ...) چندين نقش را در سيستم بازي مي کند ؟
  - آيا چندين موجوديت (کاربر ، سيستم ، فرآيند و ...) يک نقش يکسان را در سيستم بازي مي‌کنند ؟

USE CASE DIAGRAM

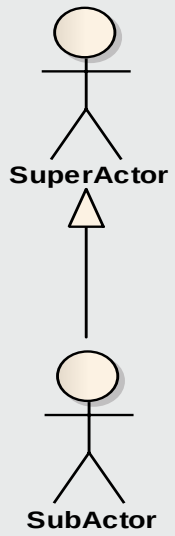
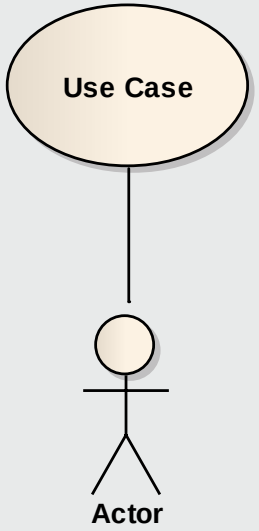
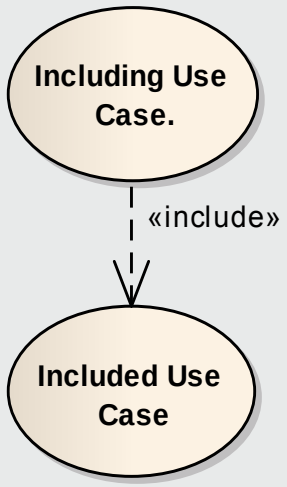
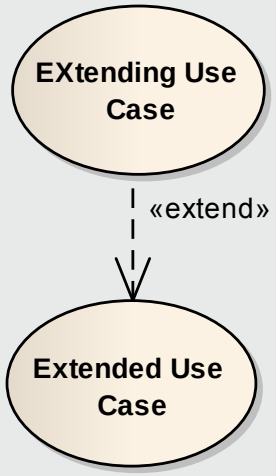
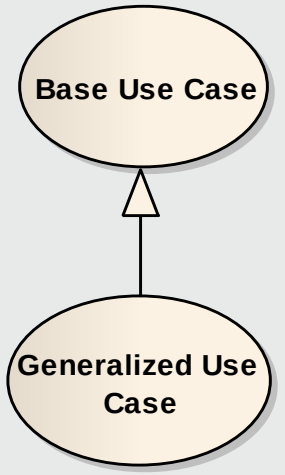
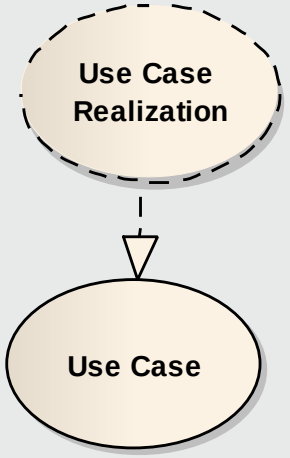
- توجه به سوالات زیر به شناسایی use case های سیستم کمک می‌کند:
  - اهداف actor ها چیست؟
  - چه پیش شرط هایی قبلی از شروع این اهداف باید وجود داشته باشند؟
  - چه تنوعی در تعاملات actor ها ممکن است؟
  - وظایف اصلی ای که actor از سیستم انتظار دارد چیست ؟
  - آیا actor بر روی داده هایی در سیستم عملیات ایجاد ، ذخیره ، تغییر ، حذف ، یا خواندن را انجام میدهد ؟
  - آیا actor نیاز دارد که سیستم را در مورد تغییرات خارجی ناگهانی آگاه کند ؟
  - آیا actor نیاز دارد که در مورد اتفاقات معینی در سیستم مطلع شود ؟
  - آیا actor عمل شروع یا خاتمه سیستم را انجام میدهد ؟

USE CASE DIAGRAM

- همانطور که گفته شد، از نمودار use case برای بیان نیازمندهای کارکردی سیستم استفاده می‌شود.
- با توجه به اینکه فرآیند استخراج نیازمندی یک فرآیند iterative است، تمام actor ها و usecase ها به صورت یک باره در سیستم شناخته نمی‌شوند.
- به عبارتی ابتدا primary actor های سیستم شناخته می‌شود و سپس secondary actor ها.
  - primary actor ها: کسانی که مستقیما و غالبا از سیستم استفاده میکنند. (از system function ها)
  - secondary actor ها: کسانی هستند که از سیستم پشتیبانی می کنند تا primary actor ها بتوانند کاری که می خواهند را انجام بدهند.
    - برای مثال system actor، **time actor**

USE CASE DIAGRAM

- ارتباطات در نمودار use case

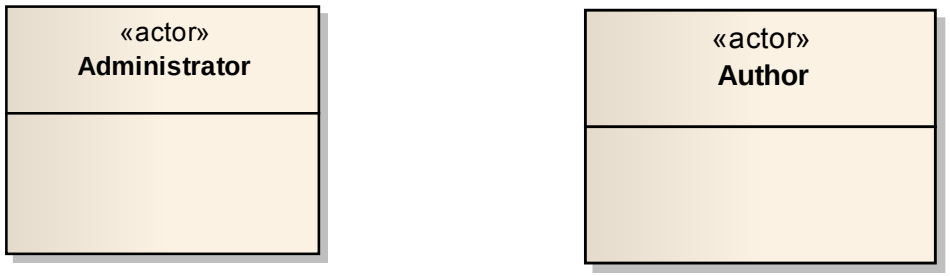
Actor - Actor	Actor - UseCase	Use case – Use Case			
Generalization	Association	Include	Extend	Generalization	Realization
					

USE CASE DIAGRAM- CASE STUDY 1(CMS)

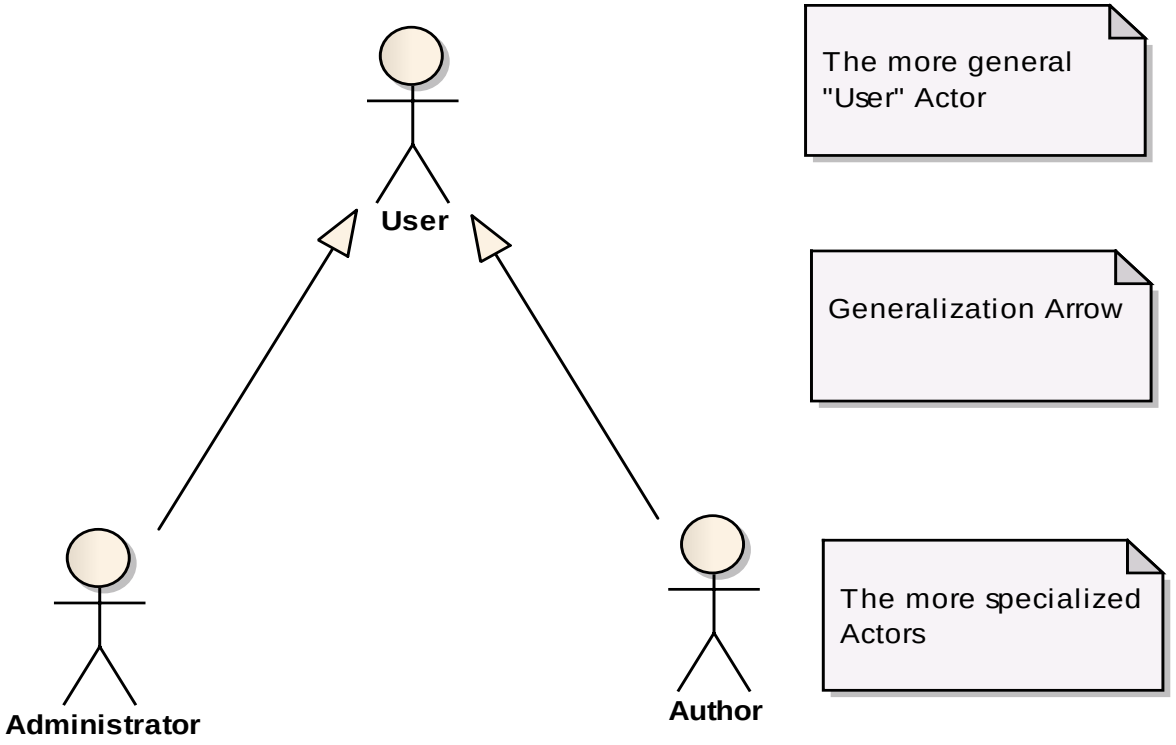
- **weblog content management system (CMS)**
  - **Req1 :** The content management system shall allow an administrator to create a new blog account, provided the personal details of the new blogger are verified using the author credentials database.
  - **Req2:** The content management system shall allow an administrator to create a new personal Wiki, provided the personal details of the applying author are verified using the Author Credentials Database.

USE CASE DIAGRAM- CASE STUDY 1(CMS)

شناسایی actor ها

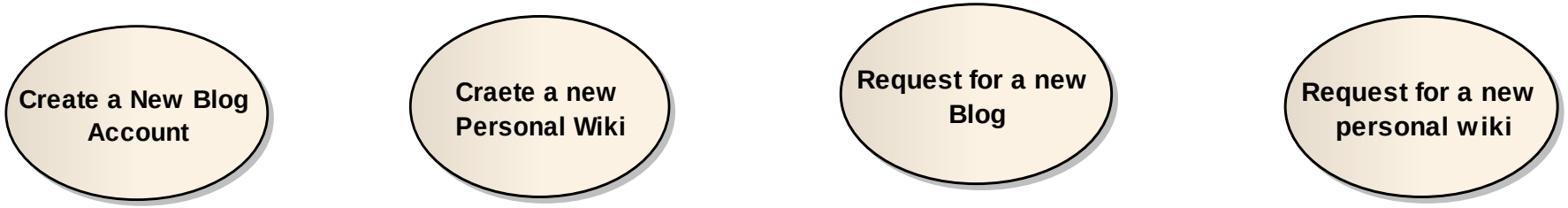


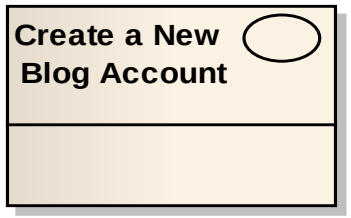
تشخیص روابط بین actor ها



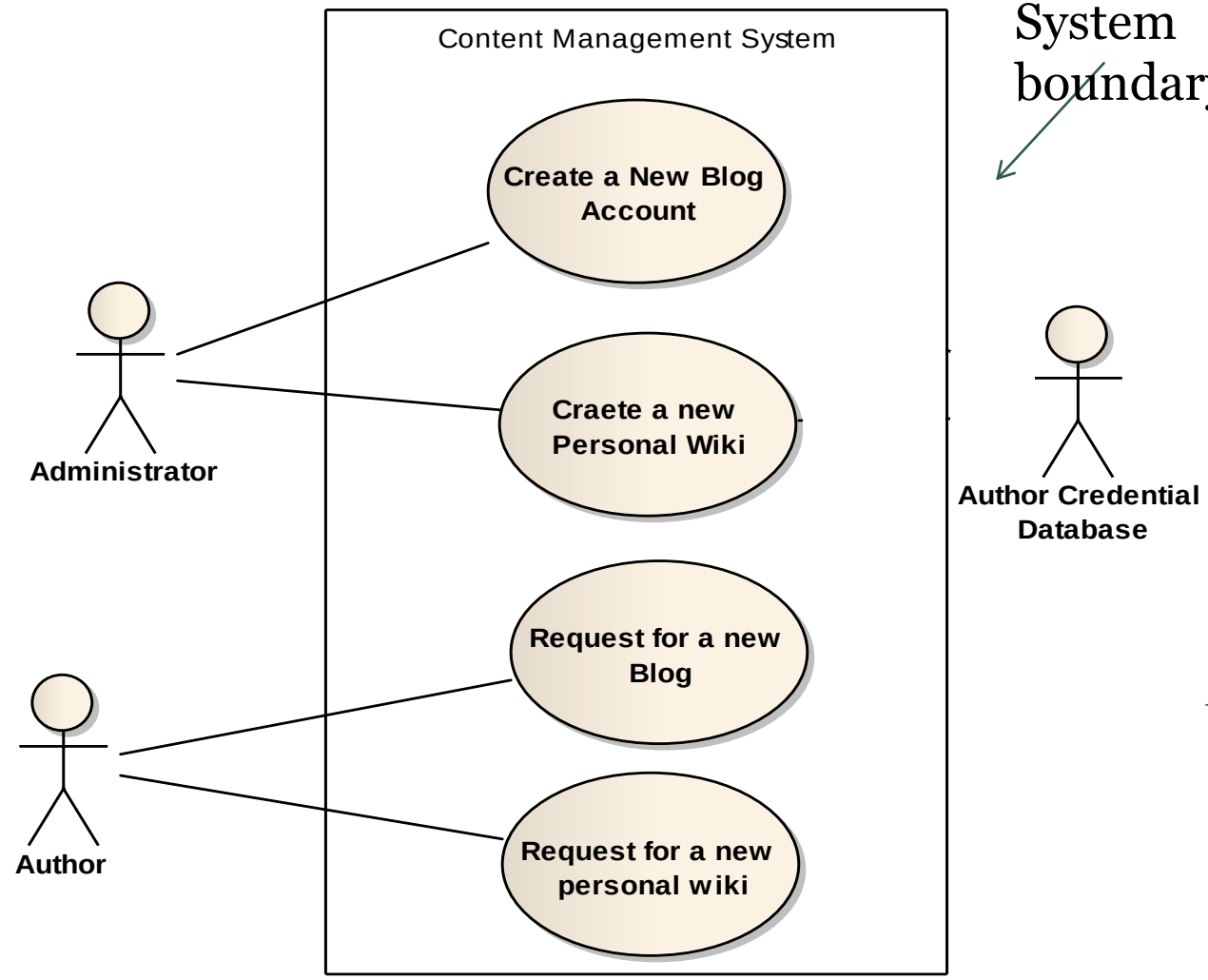
USE CASE DIAGRAM- CASE STUDY 1(CMS)

شناسایی use case ها





USE CASE DIAGRAM- CASE STUDY 1(CMS)



- مشخص نمودن ارتباطات بین actor ها و use case ها
  - مشخص می‌کند که یک actor در جریان یک use case شرکت دارد (اطلاعات را وارد میکند، از اطلاعات استفاده می‌کند، عملیاتی را trigger می‌کند و ....)
  - در نمایش این ارتباط استفاده از جهت توصیه نمیشود.
- مشخص نمودن مرزهای سیستم
  - به وضوح نشان می‌دهد actor ها در خارج از سیستم قرار دارند.

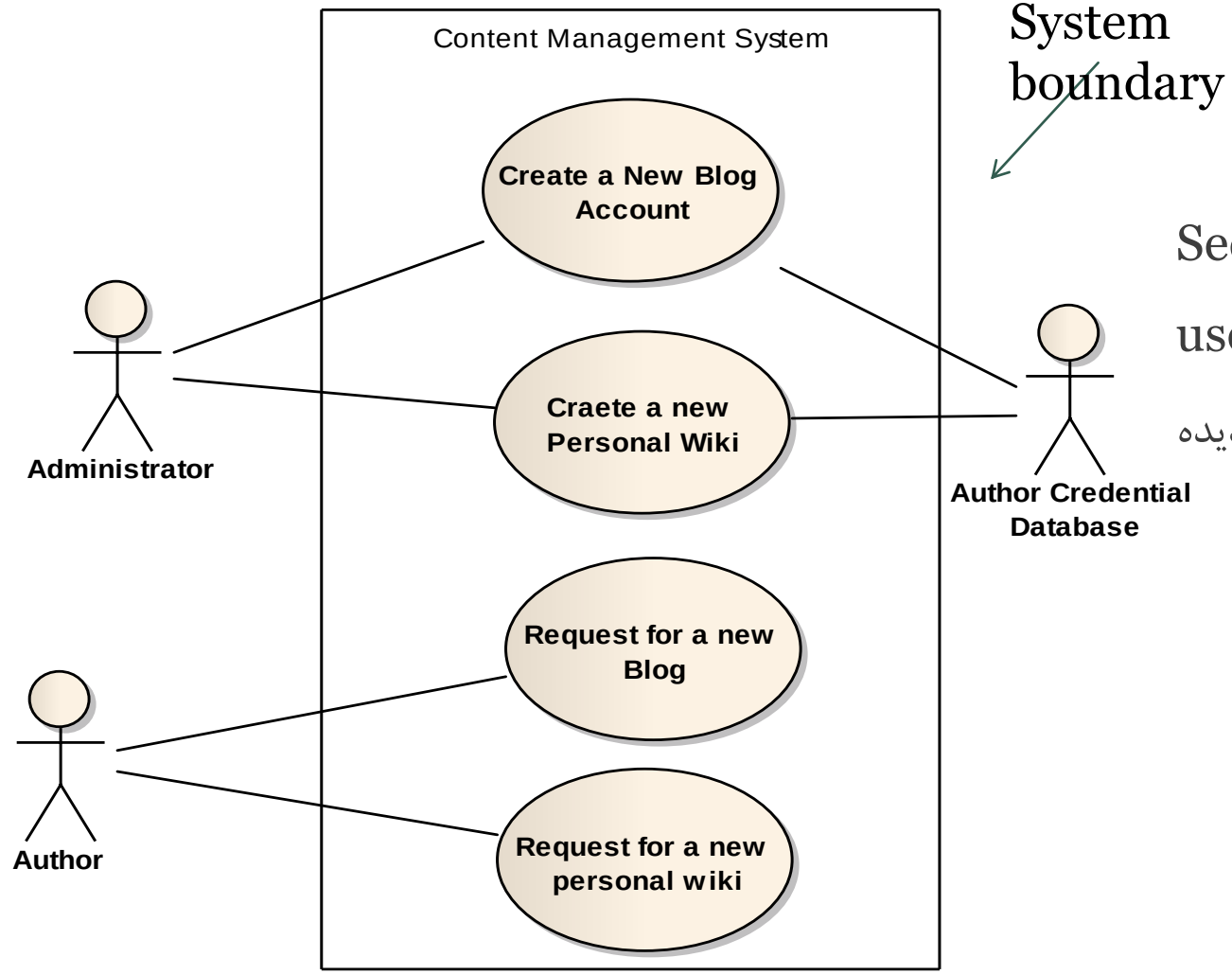
Use case diagram- Case Study 1(CMS) – Use case description

Use case name	Create a new blog account	
Related Requirements	Req1	
Goal In Context	A new blog account should be created by Administrator for a new or existing author who has sent a request for it.	
Preconditions	The system is limited to recognized authors and so the author needs to have appropriate proof of identity.	
Successful End Condition	A new blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Administrator	
Secondary Actors	Author Credentials Database.	
Trigger	The Administrator asks the CMS to create a new blog account.	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4	The author's details are verified using the Author Credentials Database.
	5	The new account is created.
	6	A summary of the new blog account's details are emailed to the author.
Extensions	Step	Branching Action
	4.1	The Author Credentials Database does not verify the author's details.
	4.2	The author's new blog account application is rejected.

USE CASE DIAGRAM- CASE STUDY 1(CMS)

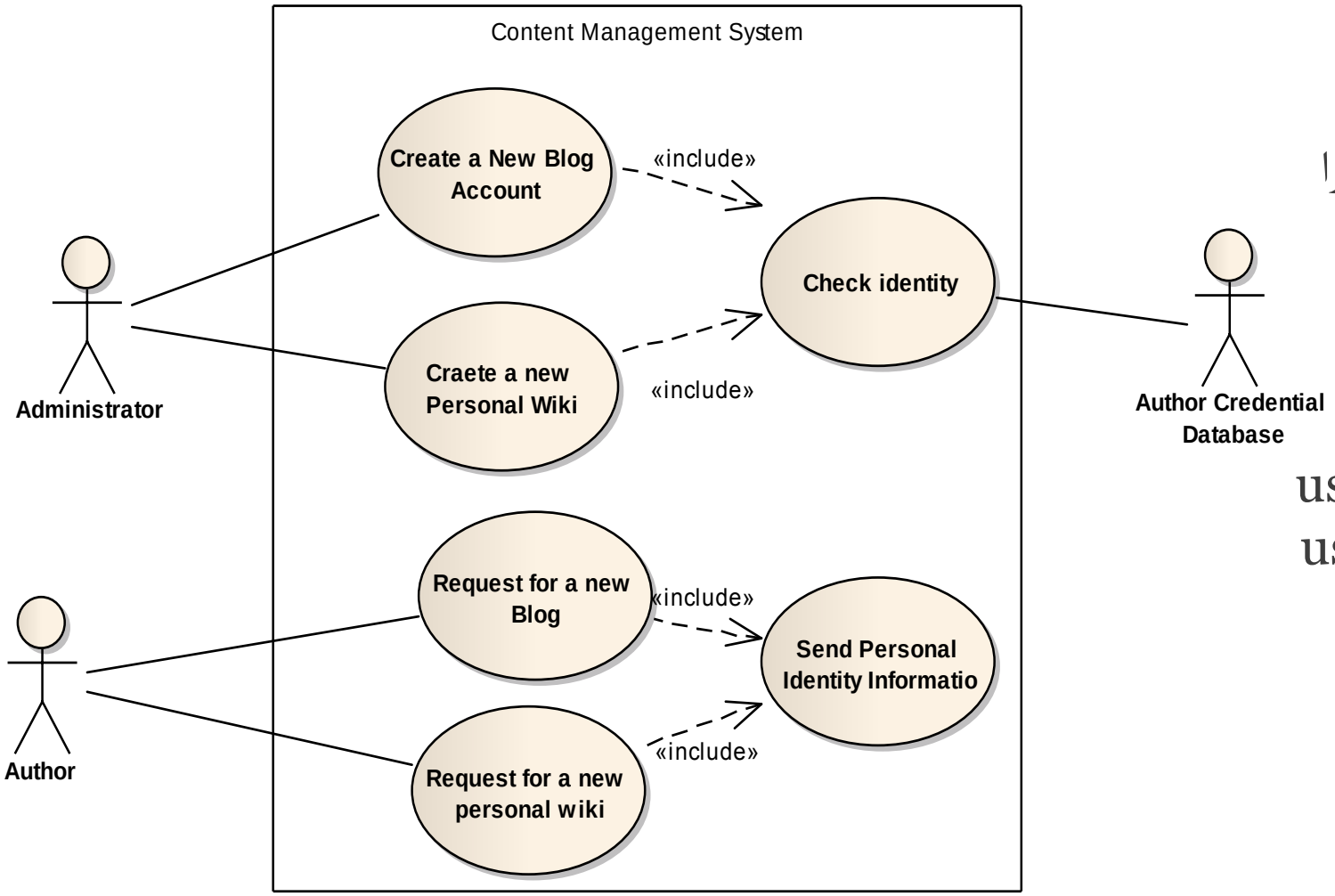
- علاوه بر صفات مطرح شده (دو اسلاید قیل) در شرح use case، می‌توان صفات زیر را نیز وارد نمود:
  - Priority
  - Frequency of use
  - Channel to use
  - When available
  - Status

USE CASE DIAGRAM- CASE STUDY 1(CMS)



- افزودن جزئیات بیشتر به مدل
  - Author Credentail Database یک Secondary actor است که پس از جزئی نمودن و نوشتن توصیف use Create a new blog account case شناسایی گردیده است.

USE CASE DIAGRAM- CASE STUDY 1(CMS)



- رابطه‌ی include بین usecase ها
  - یک use case رفتار یک use case دیگر را شامل می‌شود.
  - حذف redundancy بین use case ها
  - reusability را افزایش می‌دهد.
  - بیان کننده‌ی deligation است. (یک use case بخشی از وظایف خود را به use case دیگر واگذار می‌کند.)

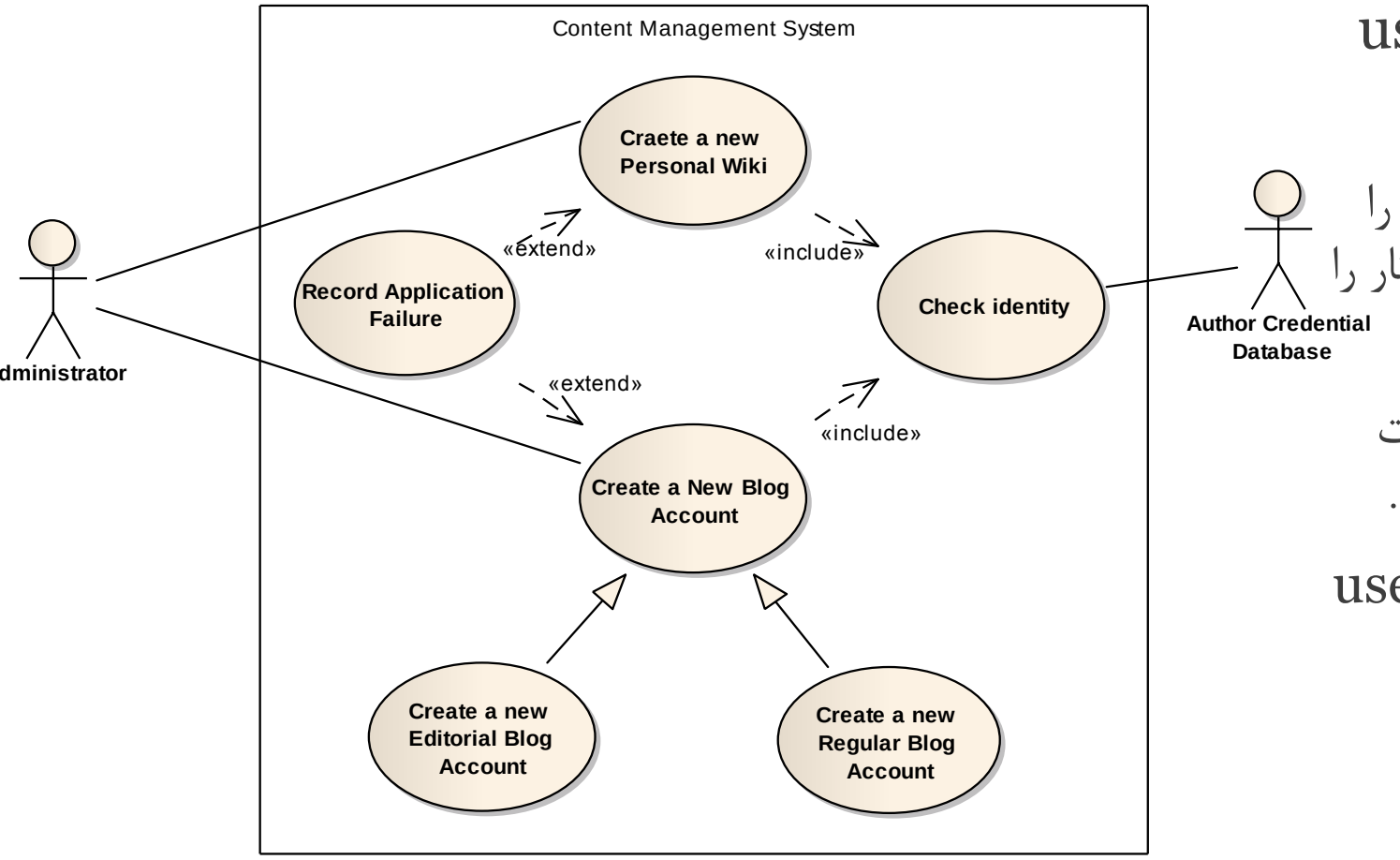
Use case diagram- Case Study 1(CMS) – Use case description

Use case name	Create a new blog account	
Related Requirements	Req1	
Goal In Context	A new blog account should be created by Administrator for a new or existing author who has sent a request for it.	
Preconditions	The system is limited to recognized authors and so the author needs to have appropriate proof of identity.	
Successful End Condition	A new blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Administrator	
Secondary Actors	Author Credentials Database.	
Trigger	The Administrator asks the CMS to create a new blog account.	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	The Administrator selects an account type.
	3	The Administrator enters the author's details.
	4 **include::CheckIdentity**	The author's details are checked.
	5	The new account is created.
	6	A summary of the new blog account's details are emailed to the author.
Extensions	Step	Branching Action
	4.1	The Author Credentials Database does not verify the author's details.
	4.2	The author's new blog account application is rejected.

Use case diagram- Case Study 1(CMS)

Use case name	Check Identity	
Related Requirements	Req1 , Req 2	
Goal In Context	An author's details need to be checked and verified as accurate.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	The details are verified.	
Failed End Condition	The details are not verified.	
Primary Actors	Author Credentials Database.	
Secondary Actors	-	
Trigger	An author's credentials are provided to the system for verification.	
Main Flow	Step	Action
	1	The details are provided to the system.
	2	The Author Credentials Database verifies the details.
	3	The details are returned as verified by the Author Credentials Database.
Extensions	Step	Action
	2.1	The Author Credentials Database does not verify the details.
	2.2	The details are returned as unverified.

USE CASE DIAGRAM- CASE STUDY 1(CMS)

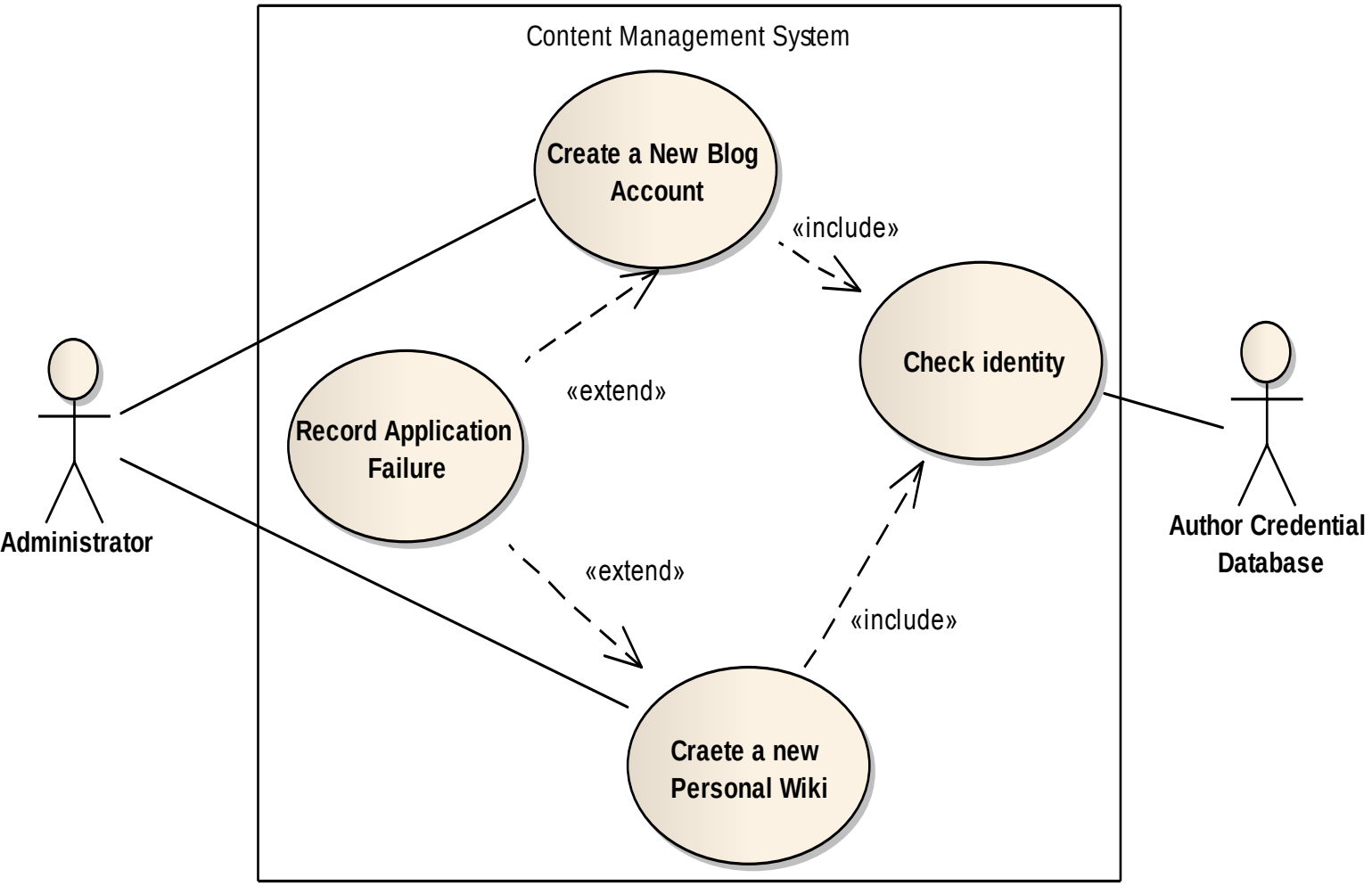


- رابطه‌ی Generalization بین use case ها
  - Sub usecase رفتار Super usecase را به ارث می‌برد و ممکن است به نحوی این رفتار را تغییر داده و override کند.
  - Sub usecase تمام وابستگی ها و ارتباطات مرتبط با Super usecas را به ارث می‌برد.
  - Sub usecase تمام گام های موجود در use case اصلی را شامل می شود.

Use case diagram- Case Study 1(CMS)

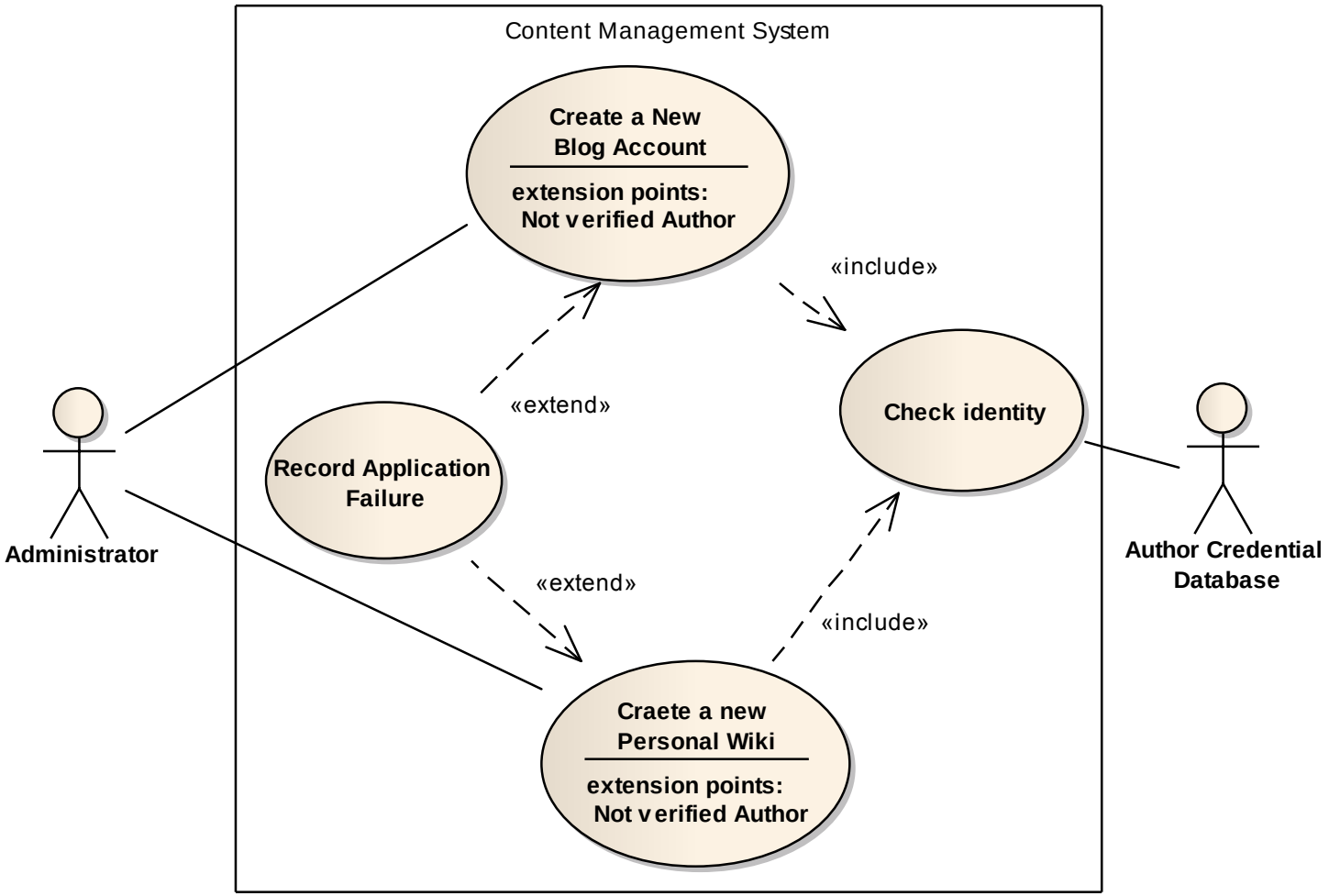
Use case name	Create a new **Editorial** blog account	
Related Requirements	Req1	
Goal In Context	A new **Editorial** blog account should be created by Administrator for a new or existing author who has sent a request for it.	
Preconditions	The author has appropriate proof of identity.	
Successful End Condition	A new **Editorial** blog account is created for the author.	
Failed End Condition	The application for a new blog account is rejected.	
Primary Actors	Administrator	
Secondary Actors	-	
Trigger	The Administrator asks the CMS to create a new blog account.	
Main Flow	Step	Action
	1	The Administrator asks the system to create a new blog account.
	2	**The Administrator selects the editorial account type.**
	3	The Administrator enters the author's details.
	4	The Administrator selects the blogs that the account is to have editorial rights over.
	5 **include::Check Identity**	The author's details are checked.
	5	The new editorial account is created.
	6	A summary of the new blog account's details are emailed to the author.
Extensions	Step	Branching Action
	5.1	**The author is not allowed to edit the indicated blogs.**
	5.2	**The editorial blog account application is rejected.**
	5.3	**The application rejection is recorded as part of the author's history.**

USE CASE DIAGRAM- CASE STUDY 1(CMS)



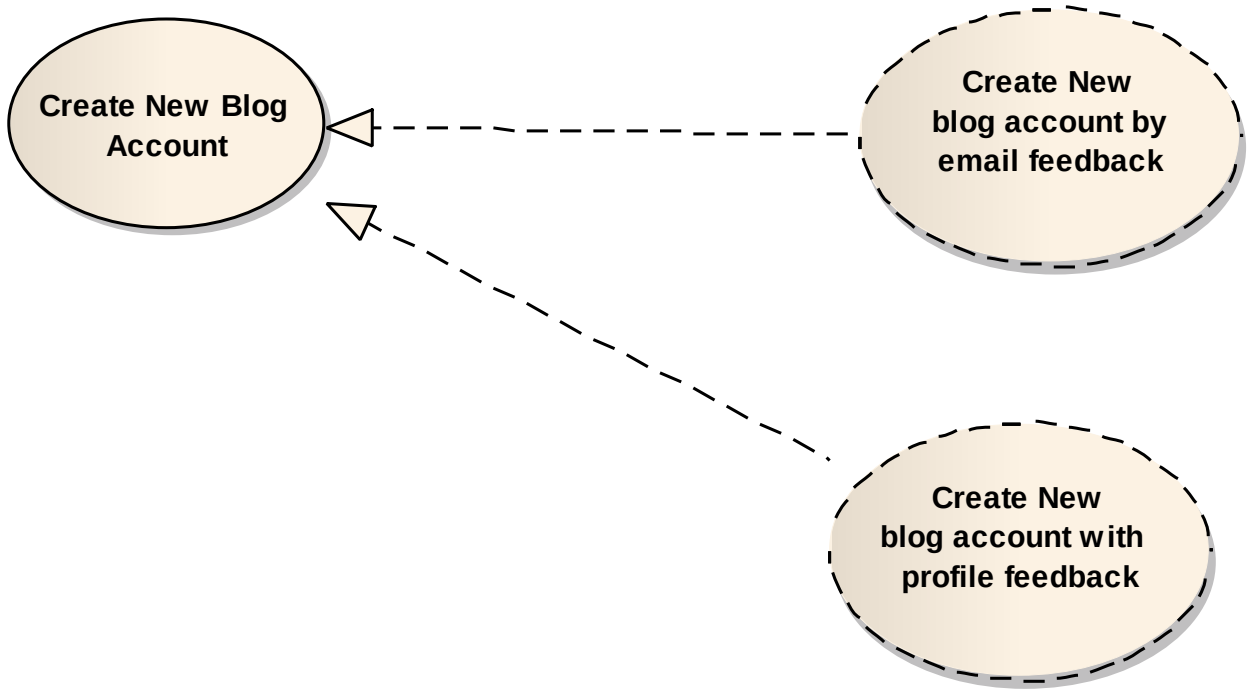
- رابطه‌ی extend بین use case ها
  - یک use case عملکرد use case دیگر را توسعه می‌بخشد.
  - به عبارتی یک use case شامل بخش‌های optional و یا exceptional یک use case دیگر می‌شود.
  - Extention ها در نقاط به خصوصی از use case اصلی اتفاق می‌افتد که به آن extension point گفته می‌شود.

USE CASE DIAGRAM- CASE STUDY 1(CMS)



- رابطه‌ی extend بین use case ها
  - یک use case ممکن است بیش از یک extension point داشته باشد.

USE CASE DIAGRAM- CASE STUDY 1(CMS)



- یک use case realiztion مشخص می‌کند که use case از طریق artifact ها و عناصری که با یکدیگر به همکاری می‌پردازند چگونه پیاده سازی می‌گردد.
- هدف از این ارتباط جداسازی یک use case از نحوه‌ی realization آن است.
- با استفاده از این ارتباط می‌توان مشخص نمود که use case را می توان به صورت های مختلف طراحی و پیاده‌سازی نمود.