UML 2.0

Genomgång av de 13 diagram-typer som ingår i UML 2.0.

AGENDA

Agenda:

- Allmänt om UML, modeller och kommunikation.
- Generell UML-notation.
- Genomgång av de 13 diagram som ingår i UML 2.0:
  - Class Diagram..... 20
  - Object Diagram..... 30
  - Package Diagram..... 34
  - Composite Structure Diagram..... 39
  - Component Diagram..... 43
  - Deployment Diagram..... 50
  - Use Case Diagram..... 55
  - Activity Diagram..... 60
  - Communication Diagram..... 66
  - Sequence Diagram..... 70
  - Interaction Overview Diagram..... 74
  - State Machine Diagram..... 78
  - Timing Diagram..... 82
- Avslutning.

Mål:

- Att visa på nyttan av att kunna kommunicera via modeller.
- Att ge exempel på hur olika modeller kan visualiseras med hjälp av UML.

ALLMÄNT OM UML

UML är en förkortning av Unified Modeling Language.

Kortfattat kan UML beskrivas så här:

- UML är ett visuellt språk för att specificera, konstruera och dokumentera ingående delar i system. UML är ett generellt modelleringsspråk i den mening att det går att använda för olika verksamhetområden och för olika tekniska plattformar.

Historik:

- UML 1.0 släpptes 1997.
- UML 2.0 finns ute i preliminär version och håller just nu på att färdigställas.
- De som mest aktivt deltog tidigare i framtagande av UML var:
  - Grady Booch (Booch)
  - Jim Rumbaugh (OMT)
  - Ivar Jacobson (Objectory)
- Rational ansvarade inledningsvis för UML men lämnade senare över ansvaret till OMG (Object Management Group) där vidare specifikationsarbete sker i form av en community.

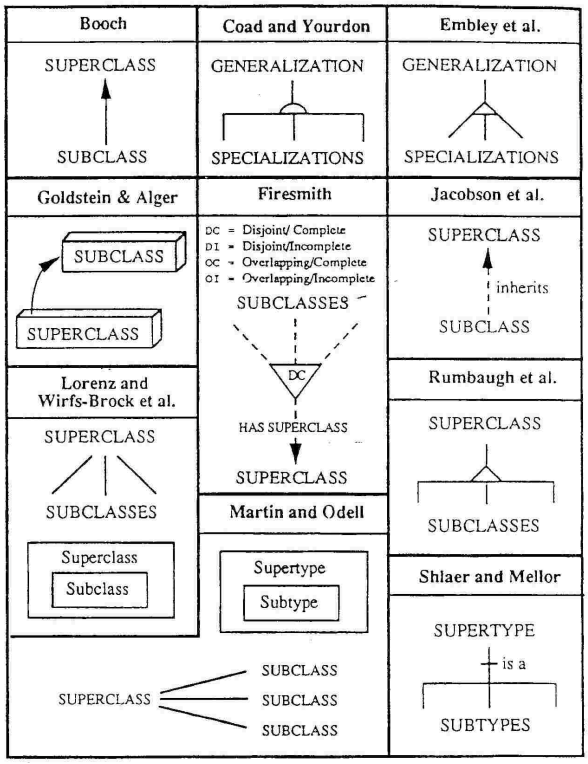
UML har lyckats etablera sig väl och blivit ”det visuella modelleringsspråket”.

Unified Modeling Language

Med ”Unified” menas enad eller likriktad.

Detta exempel på tidigare notation för arv får illustrera behovet av ett enat språk.

Nu har alla varianter i exemplet ersatts med UML:s notation för arv:



Unified Modeling Language

Med ett modelleringsspråk beskriver man modeller.

- En modell är en förenklad beskrivning av något.
- Modeller används för att beskriva en verklighet eller ett system utifrån en eller flera synvinklar. Varje sådan synvinkel har ett begränsat antal saker som man vill visa och fokusera på.
- En verksamhet kan beskrivas i form av modeller. Att ta fram sådana modeller kallas verksamhetsmodellering.
- En programvarudesign kan beskrivas i form av modeller.

Modeller och kommunikation

Modeller används för att kommunicera med andra människor. Modeller används i direkt kommunikation, i designverktyg och i dokument. Exempel på dokument är:

- En kravspecifikation används för att kommunicera mellan nyttjare och utvecklare.
- En designspecifikation är till för kommunikation mellan designers och implementatörer.
- En designbeskrivning är till för kommunikation med de personer som i framtiden ska underhålla och vidareutveckla systemet.

Modellering är en mental process som sker inne i huvudet på den som utför det.

- Den inleds med att man samlar in information genom att läsa in sig på ett område och/eller intervjua personer.
- Utifrån en given målbeskrivning tar man fram många alternativa, informella modeller och designlösningar som helt eller delvis uppfyller målen.
- När de informella modellerna och designlösningarna, eller delar av dem, börjar kännas stabil ska den diskuteras, förbättras, kompletteras och dokumenteras.

Det talade och skrivna språket är ett utmärkt medel för att diskutera och dokumentera modeller och designlösningar.

Användandet av visuella symboler snabbar ofta på informationsutbytet mellan personer, speciellt om man har olika referensramar.

Visuella symboler

Fyrkanter, streck, pilar och andra symboler fungerar ofta utmärkt för att beskriva mycket komplexa strukturer och samspel.

Nackdelen är att enkla symbolerna kan betyda allt mellan himmel och jord. Därför måste man i ord eller text i varje enskilt fall beskriva vad man menar.

Använd gärna helt fri notation vid diskussioner med kollegor.

För mer bestående beskrivningar bör man använda en mer formell grafisk notation, och då är UML:s symbolflora att rekommendera.

Unified Modeling Language

Några egenskaper hos UML som modelleringsspråk:

- UML är ett visuellt modelleringsspråk.
- UML består att en stor uppsättning element och regler för hur elementen kopplas samman.
- Elementen används i diagram. 13 olika diagramtyper finns identifierade i UML 2.0, tidigare var det 9 stycken.
- Elementen har en definierad syntax och semantik, vilket gör att det kan kallas för ett språk.
- UML är ett semi-formellt språk. Man har valt ett intuitivt och pragmatiskt språk före ett rent formellt språk.

UML är utökningsbart

Designmål vid framtagandet av UML vara att det skulle vara flexibelt, utökningsbart, mångsidigt och generellt för att kunna utgöra en grund för alla typer av modelleringsbehov som finns vid utvecklandet av system.

- Inför lanseringen av UML 1.0 (OOPSLA -96) var svaret på alla frågor om anpassningar/utökningar: ”Använd stereotyper”.
- I UML 2.0 finns flera olika möjligheter. I första hand används stereotyper, profiler och OCL.
- Även utökningsmekanismerna är en del av UML-standarden.
- Via profiler skapas dialekter av UML som kan användas för anpassningar inom olika verksamhetsområden eller teknikområden (typ J2EE/EJB).
- OCL, Object Constraints Language, är ett icke-visuellt språk där formella villkor kan uttryckas.

UML och verktygsstöd

UML är inte ett verktyg för att skapa programvarusystem. Det är ett visuellt språk för kommunikation, modellering, specifikation och definition av system. Stor vikt har dock lagts för att erbjuda möjligheten att konstruera verktyg som stödjer UML.

- Historiskt sett finns en koppling mellan UML och de objektorienterade språken. Kodgenerering och kodvisualisering är därför de område där verktygsstödet har funnits längst.
- XMI är en standard för utbyte av modeller mellan verktyg. För att kunna hantera ett så pass generellt och flexibelt språk som UML har man definierat olika nivåer (”Compliance Levels”) som definierar vad som ska vara möjligt att exportera/importera. Level 0 - 3 finns definierade i UML 2.0 där level 3 representerar hela UML.

UML-modell kontra diagram i verktyg

När man bygger system med hjälp av designverktyg ska man i första hand tänka på den modell som skapas. För att visualisera modellen använder man diagram. Diagrammen är sällan det centrala i ett verktyg utan det är den modell som skapas när man arbetar med diagrammen.

- En UML-modell består av element som paket, klasser och associationer.
- Ett UML-diagram är en representation av en del av UML-modellen.
- När man arbetar med ett designverktyg baserat på UML är det lämpligt att man fokuserar på den modell som tas fram. Strukturskapande element lägger man ofta till i en ren trädstruktur, exempelvis paket och klasser.
- Innehållet i de olika diagram man skapar hämtas sedan från denna trädstruktur, ofta genom ”drag-and-drop”.
- Kopplingar mellan element (exempelvis associationer) är oftast enklast att hantera i diagram. Även om de lagts till i ett diagram kommer de att ingå i den bakomliggande modellen.

Genomgående exempel: Fyra i rad

I materialet förekommer en del exempel som är kopplade till utvecklandet av ett spel. Spelet heter **Fyra i rad** och är utvecklat i J2ME (Java Micro Edition) vilket gör att det går att köra i handdatorer och mobiltelefoner.

Bilden till höger visar hur spelet ser ut i en mobiltelefonsemulator.

Designen är skiktad i ett presentationsskikt och ett modellskikt och en handfull designmönster används i designen.

Genom att byta presentationsskikt går spelet att köra i J2SE (med Swing) eller i J2EE (med Servlets).

Exempel i materialet visar också på hur man kan föra diskussioner inför en vidareutveckling av spelet.

Elementen i UML

Språket UML byggs upp av ett antal element och regler för hur dessa element kopplas samman. Här är ett urval av UML 1.4-element :

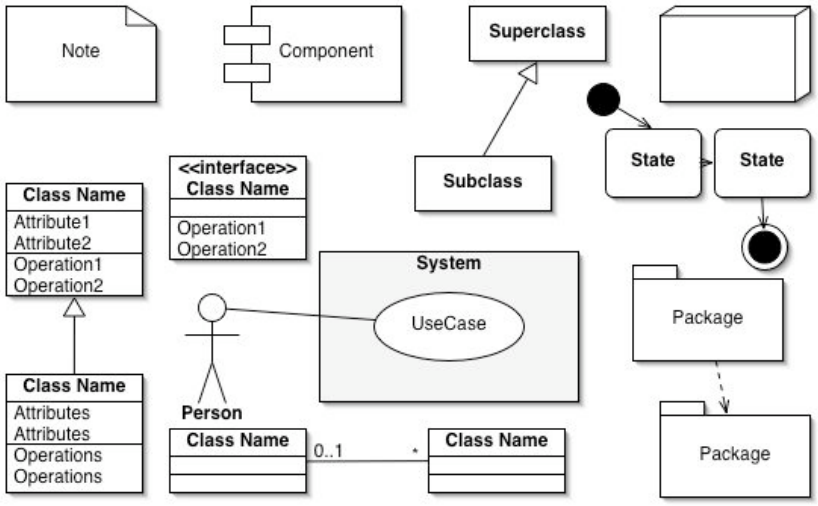


Diagram i UML 2.0

Structural Diagrams:

- Class Diagram
- Object Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram

Behavioral Diagrams:

- Activity Diagram
- Use Case Diagram
- State Machine Diagram

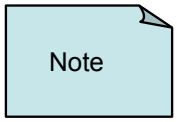
Interaction Diagrams:

- Sequence Diagram
- Communication Diagram
- Interaction Overview Diagram
- Timing Diagram

Nyheterna i UML 2.0 är: Composite Structure Diagram, Package Diagram, Interaction Overview Diagram och Timing Diagram.
Communication Diagram hette tidigare Collaboration Diagram.

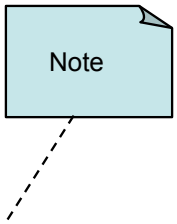
Allmän UML-notation: ”Notes”

En notering kan göras var som helst i ett diagram. Antingen som en fristående notering eller kopplad till ett element via en streckad linje.



Använd noteringar för att:

- Förklara och informera läsaren.
- Förklara det som inte är uppenbart.
- Beskriva sådant som är oklart eller ännu inte utrett.
- Skriva kodfragment eller text som beskriver ett beteende eller en begränsning.
- Det går även att skriva OCL-notation som en kommentar om det finns behov av en mer utförlig formell notation.

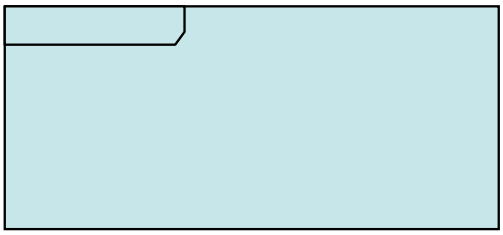


”Frames”

Ramar är en nyhet i UML 2.0. De används för att rama in grupper av samverkande element eller för att att referera till andra sådana grupper. Användningen sker huvudskligen inom två områden.

Det första är för att rama in hela diagram. Ramhuvudet kan innehålla följande specifikation av innehållet plus beskrivande text. Förkortning inom parantes:

- activity (act).
- class.
- component (cmp).
- interaction (sd).
- package (pkg)
- state machine (stm).
- use case (uc).

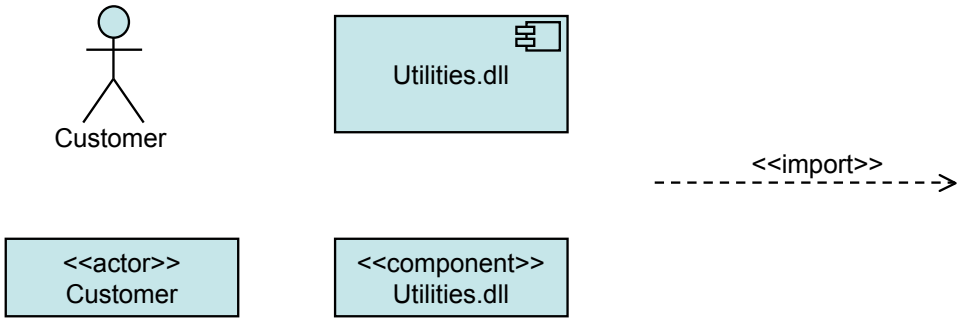


Det andra användningsområdet är i ”combined fragment frames” där man ramar in delar av diagram, exempelvis sekvensdiagram. Exempel på interaktionsoperatorer som då skrivs i ramhuvudet är: **loop** (för iterationer), **alt** (används för if-then-else), **par** (för parallella flöden). Övriga interaktionsoperatorer är: **seq**, **opt**, **break**, **strict**, **critical**, **neg**, **assert**, **ignore** och **consider**.

Stereotyper

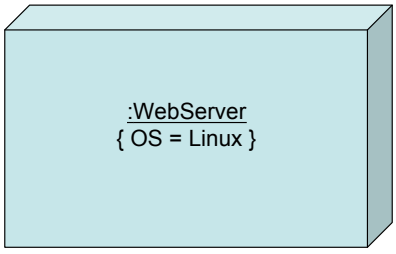
Stereotyper används för att förändra eller specificera innebörden av ett befintligt UML-element.

- Det går alltså att rita en sorts UML-element men låta det representera en annan sorts element. Det går också att skapa en ny innebörd för ett element.
- Dubbelhakar anger att det är en stereotyp. Exempel: <<component>>.
- Alternativt kan en grafisk symbol användas. Exempel:



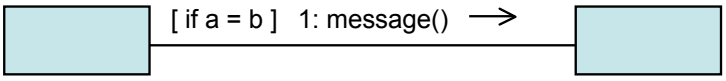
”Constraints”

- Begränsningar skrivs inom krullparanteser: { }
- Det finns inget som hindrar att ”notes” används om det är tydligare för läsaren.



”Expressions”

- Uttryck skrivs inom hakparanteser: [ ]
- Det finns inget som hindrar att ”notes” används om det är tydligare för läsaren.
- Uttryck kan skrivas som pseudo-kod, ren text eller OCL.



”CLASS DIAGRAM”

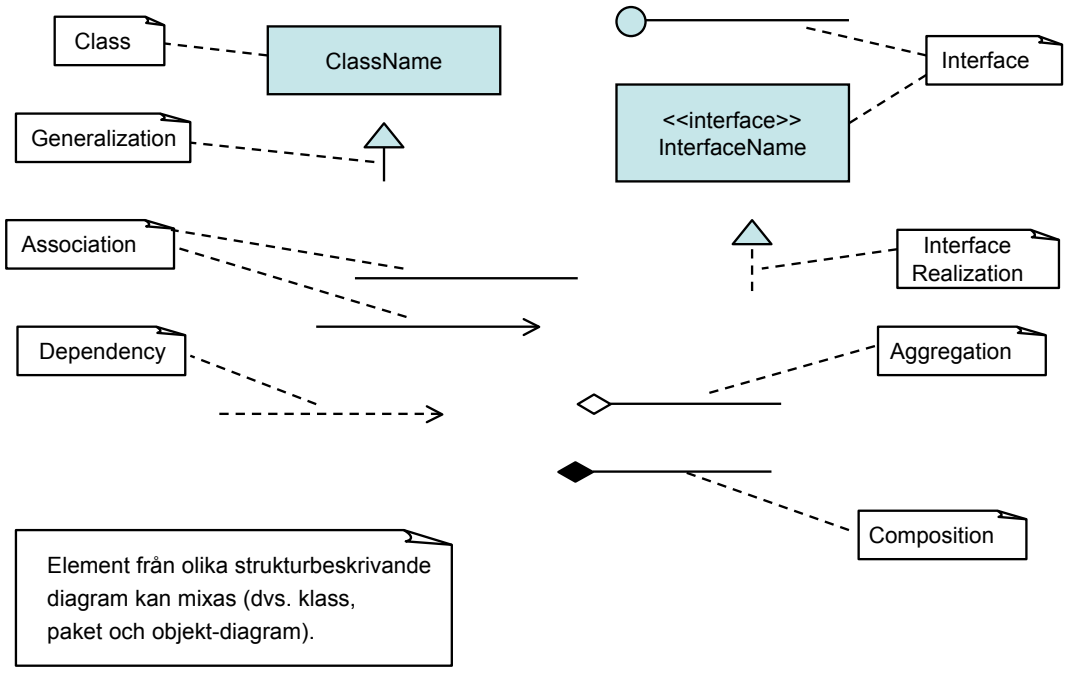
Klassdiagrammet är ett strukturbeskrivande diagram som används för att visa en statisk bild över modell-element och deras inbördes relationer. Exempel är klasser, klassers innehåll och relationer mellan klasser.

- Klassdiagram är det mest centrala digrammet i UML (compliance level 0). De mekanismer som används för klasser fungerar ofta på samma sätt för andra UML-element.
- Ofta utgår man från en klass när man vill byta innebörd med hjälp av en stereotyp. Den grafiska notationen för en klass behåller man då.
- I tidigare versioner av UML användes klassdiagrammet till fler saker än vad som beskrivs i texten ovan. I UML 2.0 har man infört Paketdiagram och ”Composite Structure Diagram” för att ta över vissa av klassdiagrammets tidigare användningsområden.

”Class Diagram”, När...

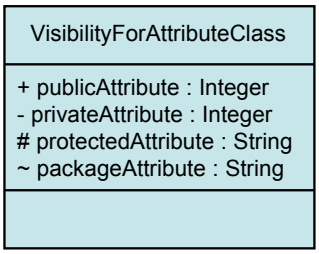
- Klassdiagram kan användas i många olika sammanhang där strukturelement och deras relation ska modelleras.
- Kodgenerering och kodvisualisering:
  Objektorienterad kod och klassdiagram har mycket gemensamt och historiskt sett är den kopplingen kärnan i de verktyg som stödjer UML. Objektorienterad kod kan visualiseras med UML och från klassdiagram kan kod genereras.
- Verksamhetsmodellering:
  Konceptuella modeller för att beskriva verksamheter är lämpliga att beskriva i form av klassdiagram.

”Class Diagram”, UML-element

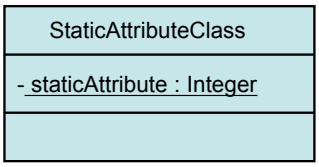


Klasser: Attribut

Exempel på synbarhet för attribut:

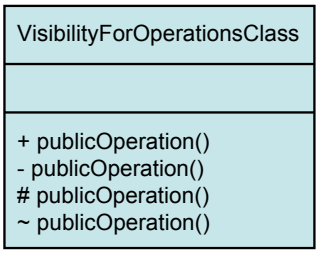


Statiskt attribut. Med statiskt attribut menas ett attribut som är kopplat till en klass och inte till instansen av en klass.

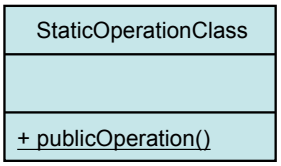


Klasser: Operationer

Exempel på synbarhet för operationer :

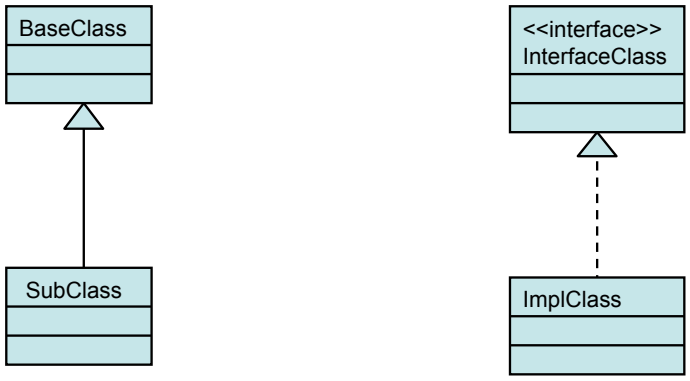


Statiskt operation. Med statiskt operation menas en operation som är kopplad till en klass och inte till instansen av en klass.



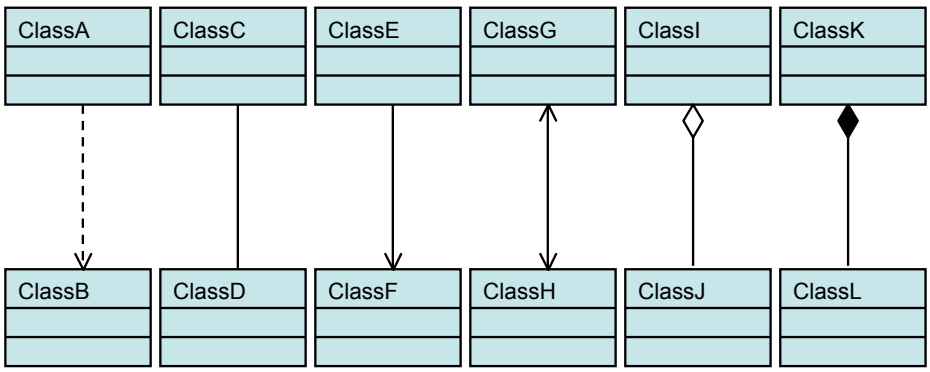
Klasser: Arv och implementation

Exempel på notation för arv (extends) och implementation.



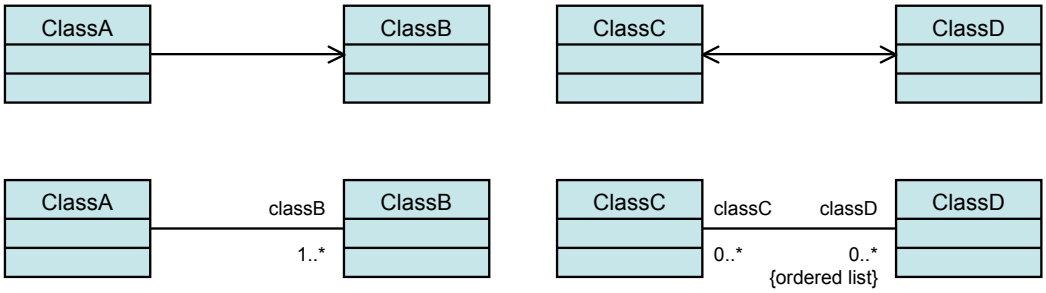
Grundregel för pilar: Den som pekar är den som utför en aktiv handling.

Klasser: Associationer



Exempel på diverse associationer mellan klasser (från vänster till höger):

- ClassA är beroende av ClassB.
- ClassC har en ospecificerad koppling till ClassD.
- ClassE känner till ClassF
- Båda klasserna känner till varandra.
- Aggregat används för att visa på en struktur med “whole/part”-relation.
- Påminner om aggregat men vid komposition kan inte ClassL existera om inte ClassK finns.



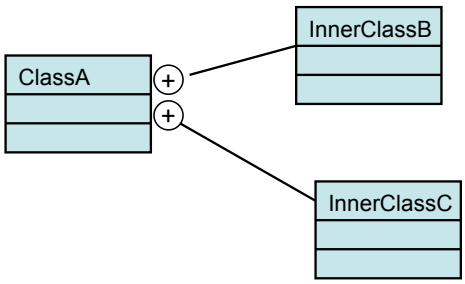
Till vänster visas två alternativa sätt för att visa att ”ClassA” känner till ”ClassB”. I det nedre alternativet ser vi att ”ClassA” har en lista med namnet classB som består av referenser av typen ”ClassB”.

Till höger visas motsvarande för en dubbelriktad koppling. Här ha man även gjort en begränsning till att ha en lista med specificerad ordning.

Kardinalitet anges med: 1, *, 0..*, 1..*, etc.

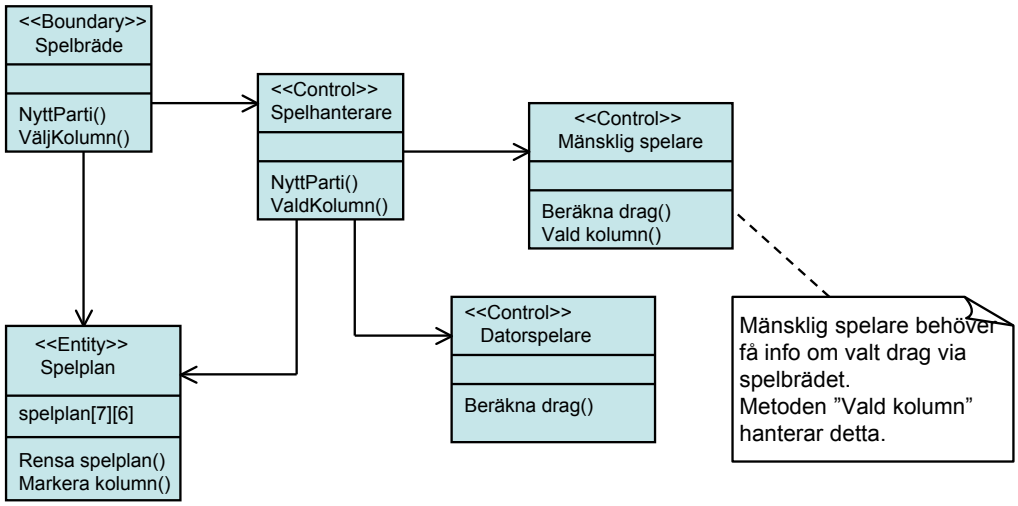
Klasser: Inre klasser

Har man behov av att modellera inre klasser så kan det göras så här:



Fyra i rad: Analysklasser

Från en tidig analysfas har detta klassdiagram hämtats. Klasserna är stereotypade som ”Boundary”, ”Control” och ”Entity”-klasser.



”OBJECT DIAGRAM”

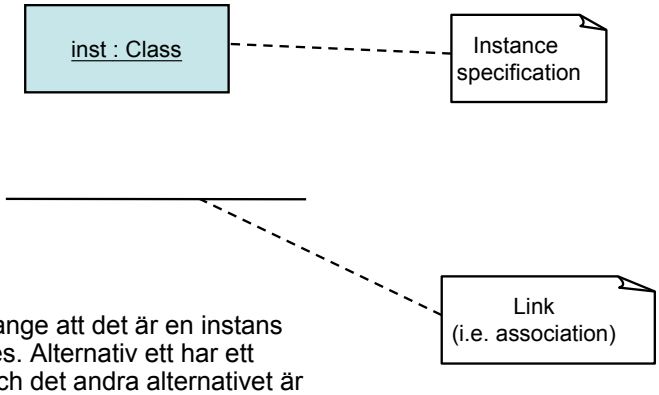
Objektdiagrammet är ett strukturbeskrivande diagram som används för att beskriva objekts tillstånd och deras relationer vid en specifikt tidpunkt. Man visar alltså en ögonblicksbild.

- Ett objektdiagram är en typ av kombination mellan klassdiagram och kommunikationsdiagram.

”Object Diagram”, När...

- Objektdiagram är speciellt användbara när man skissar på lösningar under en diskussion. Fördelen med diagrammet är att man kan diskutera kring instanser (objekt) av klasser.
- Det man får fram med objektdiagrammet är en konkret bild som är bra att använda i dialog med personer som inte är insatta i objektorientering. Ett rent klassdiagram kan för dessa personer bli för abstrakt.
- Använd inte objektdiagram i dokumentation om inte speciella skäl finns för att beskriva ett specifikt tillstånd hos en modell.
- Från ett objektdiagram går det inte att dra några slutsatser om implementationen eller runtime-strukturen.

”Object Diagram”, UML-element

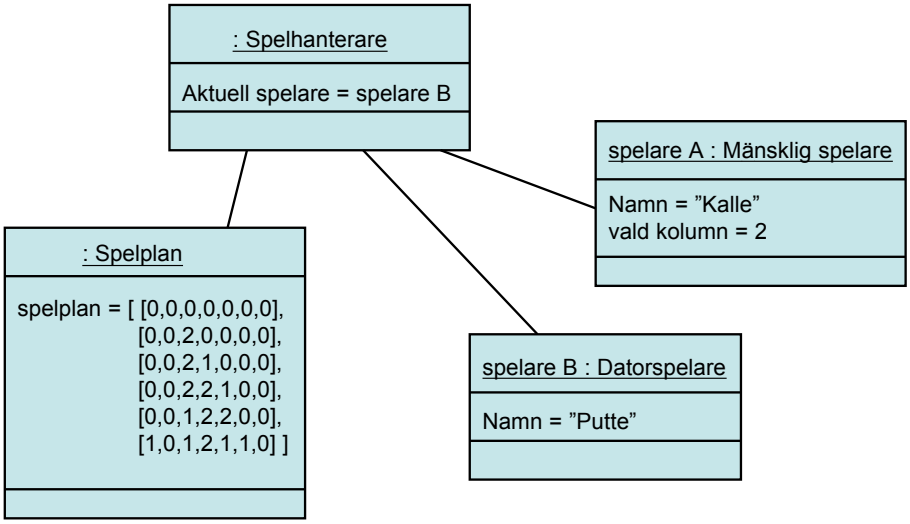


Det finns två sätt att ange att det är en instans av en klass som avses. Alternativ ett har ett namn på instansen och det andra alternativet är namnlöst:

- instanceName : ClassName
- : ClassName

Fyra i rad: Objektdiagram

Att diskutera spelet utifrån detta objektdiagram kan göra att förståelsen ökar, speciellt vad det gäller spelplanens roll.



”PACKAGE DIAGRAM”

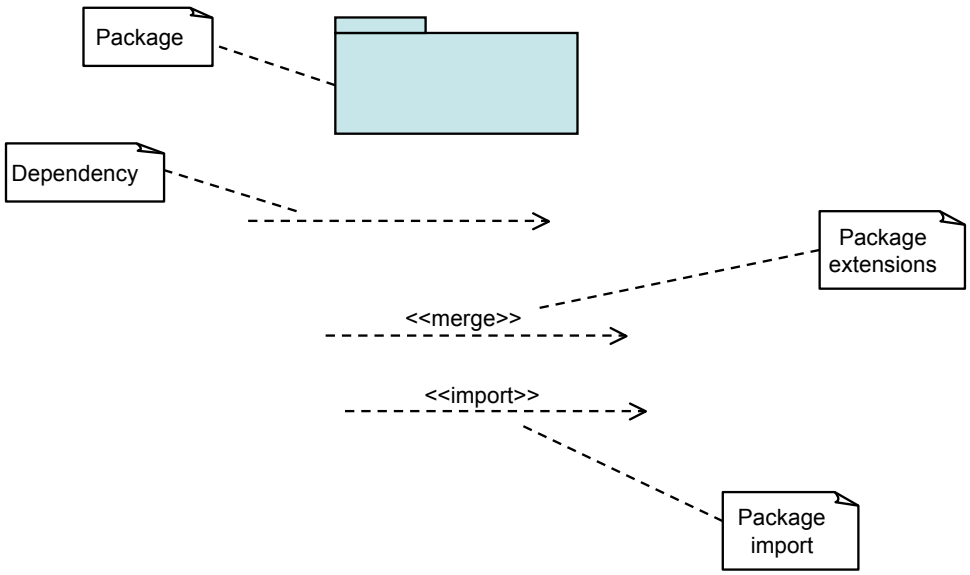
Paketdiagram är ett diagram som beskriver hur modellelement är organiserade i paket. Beroenden mellan paket beskrivs i form av ”package imports” och ”package extensions”.

- Paketdiagrammet är nytt i UML 2.0 men det har tidigare funnits som en del i andra diagram. Det som är nytt är att ett diagram som endast innehåller paket och deras inbördes relationer ska kallas Paketdiagram.
- Paket i UML är en mycket generell mekanisk och alla UML-element går att gruppera med hjälp av paket.

”Package Diagram”, När...

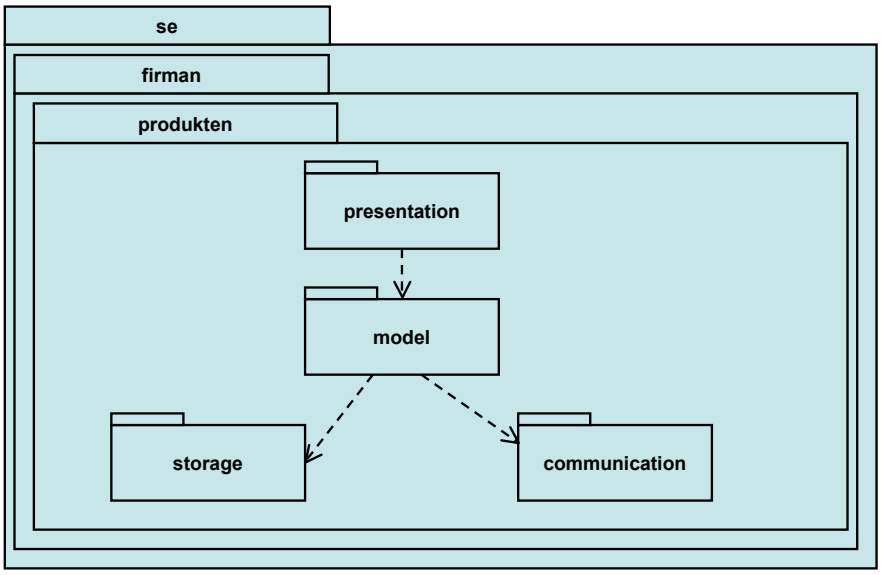
- Använd diagrammet för att skapa en övergripande beskrivning av hur element är grupperade.
- Paketstruktur för kod.
  Javas paket har en direkt koppling till den filstruktur där paketen finns. Det är lämpligt att använda paketdiagram för att visualisera denna struktur.
- Användningsfall går utmärkt att gruppera på en högre nivå. Använd paketdiagram så snart antalet användningsfall blir stort.
- Även ett “Deploymentdiagram” kan beskrivas på översta nivån med hjälp av paketdiagram.

”Package Diagram”, UML-element

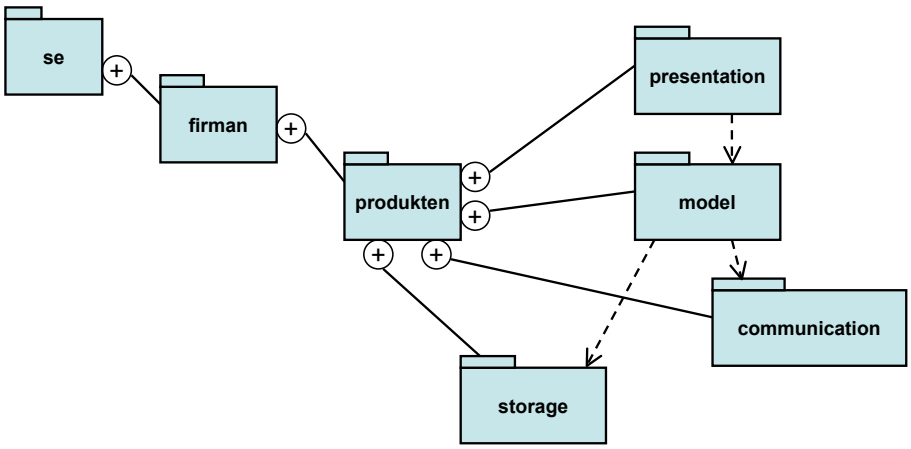


”Package Diagram”, Exempel

Paketstruktur för en produkt med namnet Produket utvecklat av det Svenska företaget Firman. Produkten är baserad på en skiktad arkitektur.



Ett alternativt sätt att visa samma information är att använda sig av symbolen för ”ingår i”. Notationen för ”ingår i” är också lämplig att använda för att visa vilka klasser som ingår i ett paket.



”COMPOSITE STRUCTURE DIAGRAM”

”Composite Structure Diagram” beskriver den interna strukturen i form av UML-element (exempelvis objekt, komponenter eller användningsfall) och hur dessa samarbetar för att uppnå ett visst syfte.

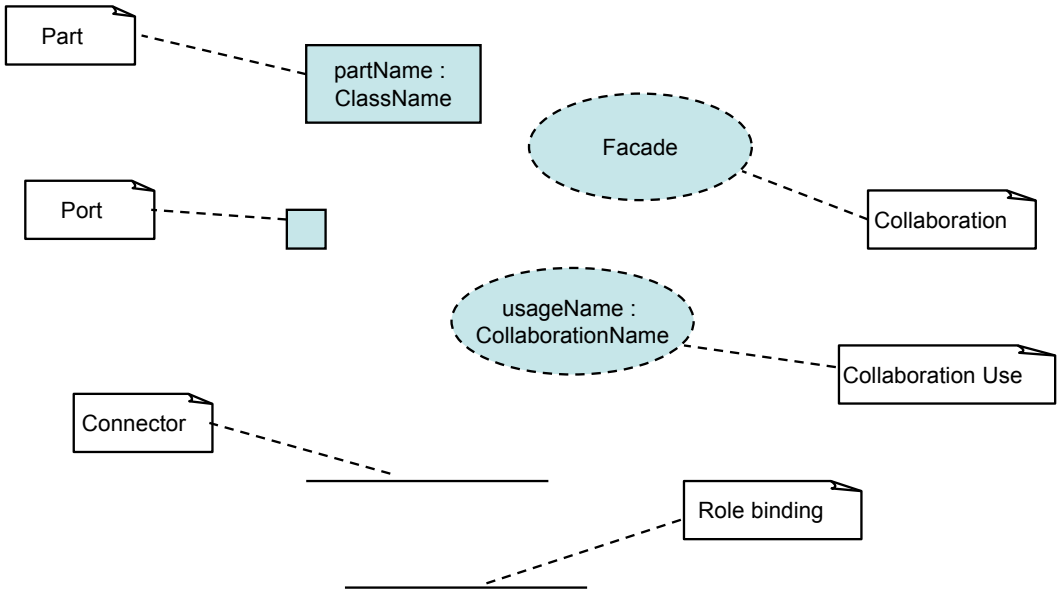
Entiteter binds samman via ”Collaborations” och man anger de roller som entiteterna har för att tillsammans utföra en eller flera uppgifter.

- Diagramtypen är ny i UML 2.0
- Tidigare har man kunnat införa motsvarande information i klassdiagram.

”Composite Structure Diagram”, När...

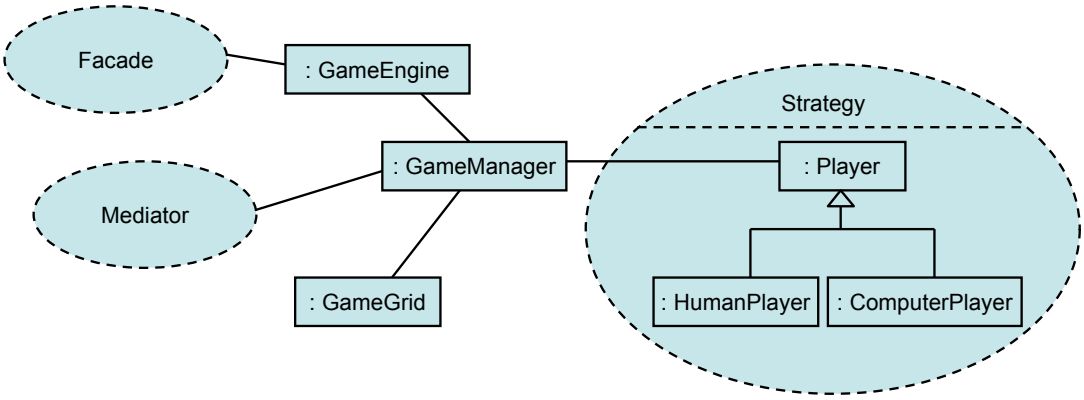
- Diagrammet är lämpligt att använda i designfasen när man arbetar med införandet av designmönster och motsvarande mekanismer.
- Använd ”Composite Structure Diagram” när man behöver beskriva rollfördelningen mellan entiteter.
- Även i arkitekturbeskrivningar finns det behov av att beskriva mekanismer och var i arkitekturen dessa hanteras.
- Eftersom diagrammet är nytt i UML så finns det inte så många exempel på användning. Behovet att beskriva mekanismer som består av samverkande delar finns i ett projekts alla faser och diagrammet verkar kunna fylla detta behov.

”Composite Structure Diagram”, UML-element



Fyra i rad: ”Composite Structure Diagram”

I spelets modellskikt använder vi tre designmönster vilket visas i nedanstående diagram med lite olika varianter av notation.



”COMPONENT DIAGRAM”

Ett komponentdiagram används för att beskriva de komponenter som tillsammans utgör en applikation, ett system eller en del av ett system.

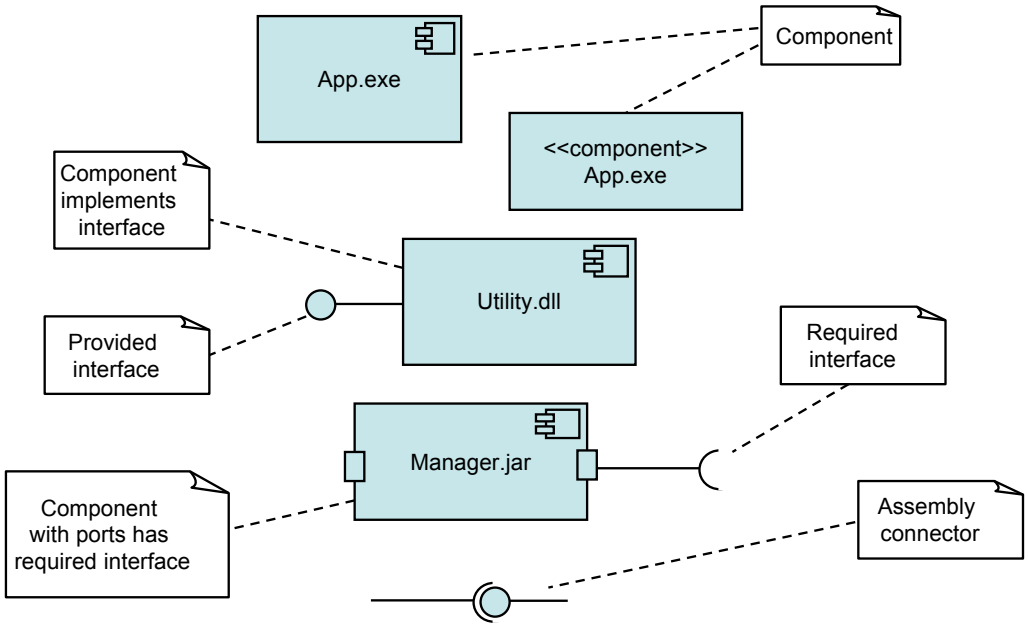
Det som beskrivs är vilka komponenterna är, hur de är organiserade samt vilka beroenden via publika interface som finns.

- Komponentdiagrammet har förändrats och kompletterats i UML 2.0.
  - Den tidigare grafiska notationen har ersatts av en rektangel som stereotypas (med <<component>> och/eller symbolen för komponent).
  - Beroenden beskrevs tidigare enbart med beroendepilen. Nu finns notation för ”required interface” som komplement till den tidigare ”provided interface”.
  - Begreppet port är infört.
- En komponent i UML 1.* kunde representera det mesta som kan lagras som fil på en dator. Nu har man infört UML-elementet ”Artifakt” som har en motsvarande definition. I UML 2.0 är en komponent något som kommunicerar via interface och som är utbytbart inom sitt område.

”Component Diagram”, När...

- Använd diagrammet för att visualisera komponentbaserade system och för att visa på vilka delar som är dynamiskt utbytbara.
- Komponentdiagram är användbara när man tar fram en övergripande arkitektur. I kombination med ”Deployment”-diagram ger det en god bild av vad som ska produceras och hur det kan driftsättas.
- Paket/klassdiagram och komponentdiagram har också en stor koppling eftersom den logiska strukturen ofta styrs av den fysiska strukturen.
- Ett komponentdiagram kan vara bra som projektplansch (“poster”) som komplement till plansch i form av klass/paket-diagram.
- Det går även att använda komponentdiagram för att visualisera verksamhetssystem.

”Component Diagram”, UML-element

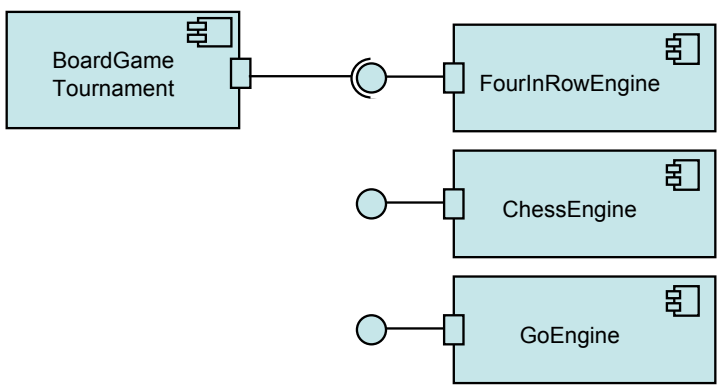


Fyra i rad: Komponentdiagram

Vill man gå vidare med programmmet Fyra i rad och bygga ett generellt system som kan hantera alla typer av brädspel kan komponentdiagrammet vara en bra start. Första steget kan vara att göra gränssnittet mot spelmotorn så generell att det fungerar för flera olika spel.

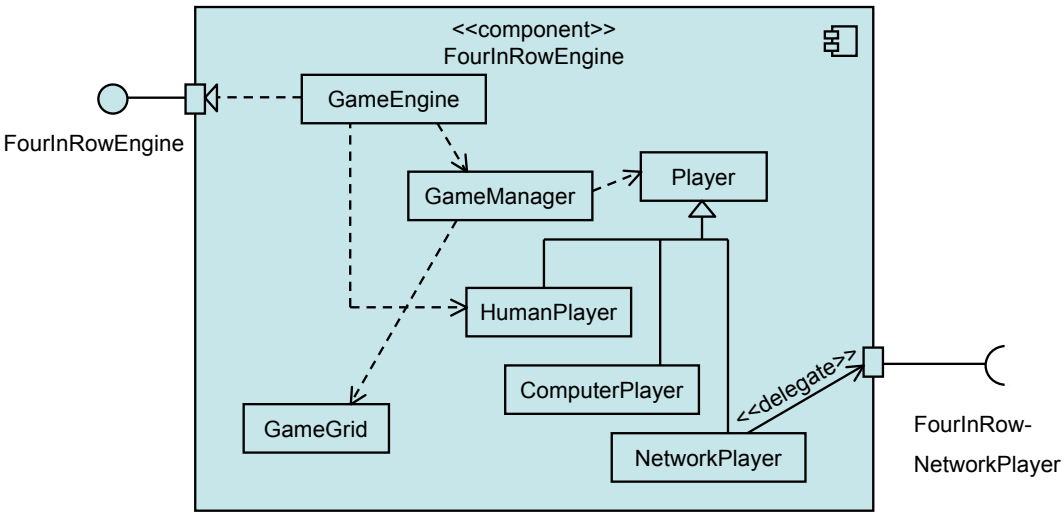
Nästa steg blir att skapa ett pluggbart användargränssnitt på motsvarande sätt.

I detta läge är typen av komponent mindre viktig och anges därför inte.



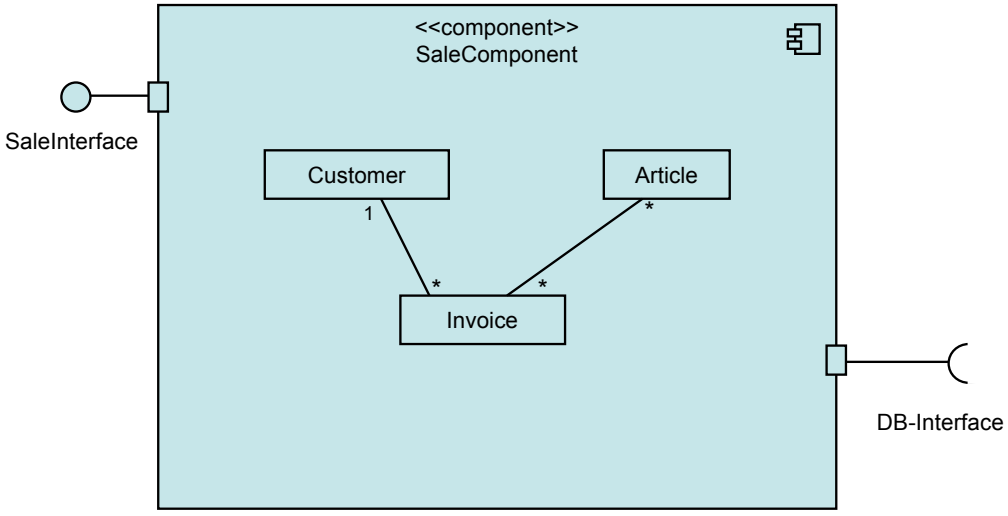
Fyra i rad: Komponentdiagram

Innehållet i en komponent kan visualiseras som ett klassdiagram.
Här visas tre helt olika alternativ för att hantera spelstrategier, en intern och två externa. Vilka alternativ som kommer att ingå i den slutliga lösningen är ännu oklart...



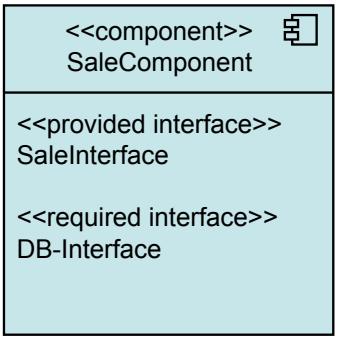
Exempel: Komponentdiagram

Ett annat sätt att förklara innehållet i ett paket är att visa en del av den konceptuella modellen. Målgruppen för en diskussion blir då inte enbart systemutvecklare. Var som hanteras inom komponenten blir tydligt presenterat.



Exempel: Komponent

Ett annat mer textorienterat sätt att beskriva en komponent är följande:



”DEPLOYMENT DIAGRAM”

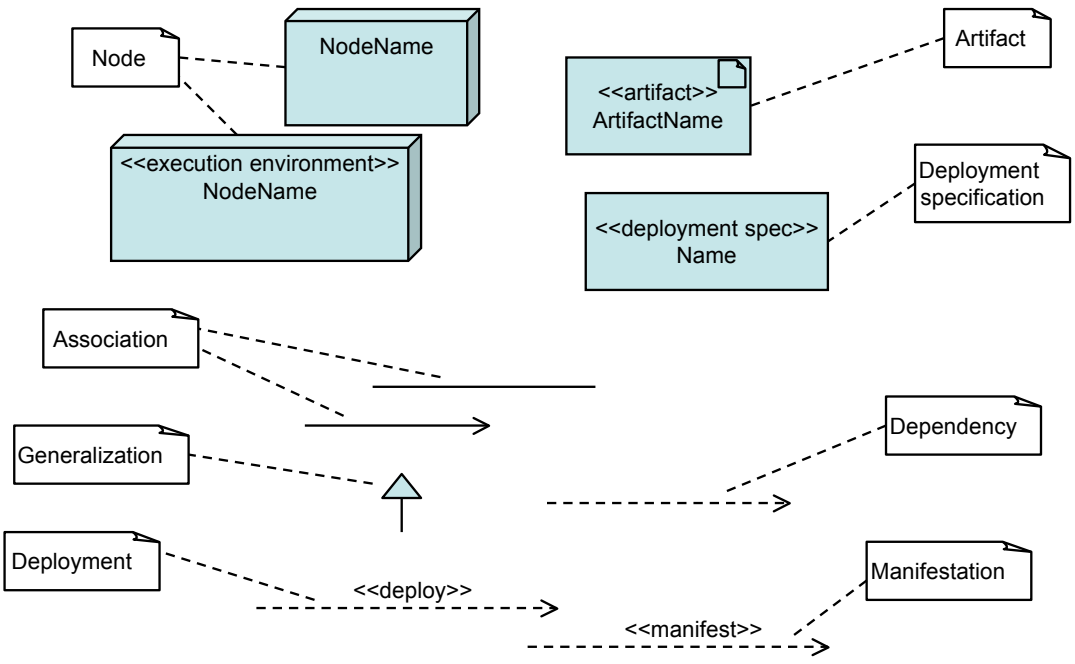
“Deployment Diagram” beskriver ett exempel på den exekverande konfigurationen för ett system. Systemets delar beskrivs som noder som är anslutna via kommunikationsvägar för att skapa nätverk av godtycklig komplexitet. En nod är något som kan exekvera program, antingen hårdvara eller en programexekverande miljö.

- Diagrammet beskriver ett ”snapshot” för en viss konfiguration av hårdvara och komponenter.
- Noder beskrivs som instanser och det finns en likhet med objektdiagram som också används för att fånga en ögonblicksbild.

”Deployment Diagram”, När...

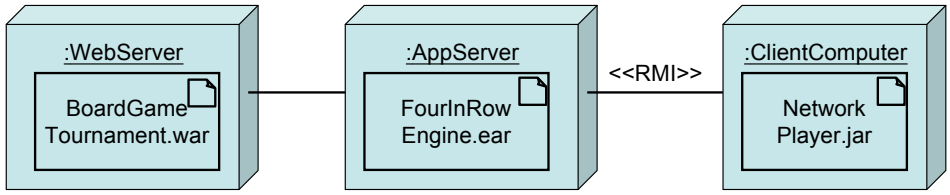
- Använd diagrammet för att beskriva hur infrastrukturen ser ut där systemet ska köras.
- Ofta kan detta diagram tas fram tidigt i projekten eftersom run-timemiljön ofta styrs av tidigare investeringar och strategibeslut utanför projektets kontroll.
- Deploymentdiagram är lämpliga att ta fram parallellt med komponentdiagram eftersom infrastrukturen ofta styr minimiantalet av komponenter som behövs.

”Deployment Diagram”, UML-element



Fyra i rad: Driftsättning

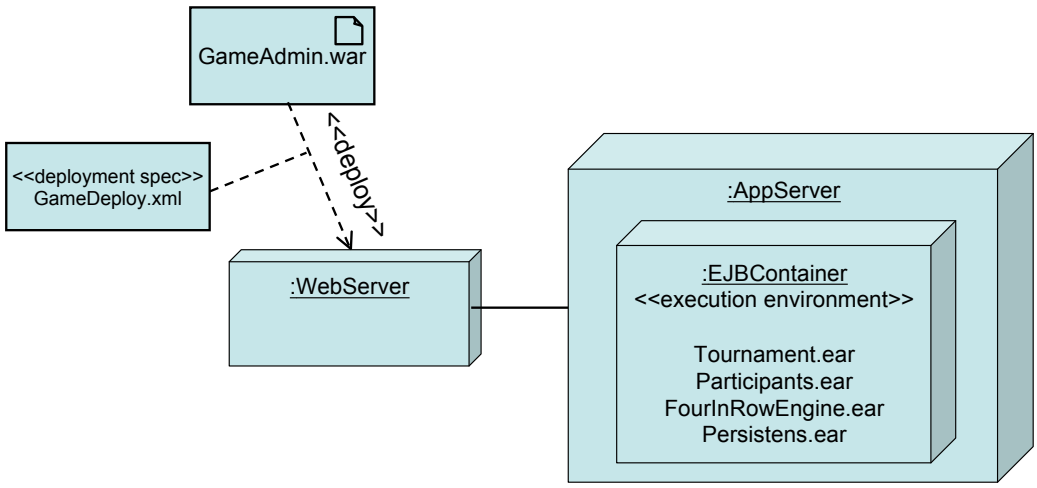
En driftsituation där en nätverksspelare ingår kan se ut så här. Nätverksspelaren har en egen applikation och utifrån det övriga webb-baserade systemet fungerar han endast som en implementation av en algoritm.



”Deployment Diagram”, Exempel

På samma sätt som man kan lista klasser i ett paket utan den grafiska symbolen för klass kan man lista komponenter i noder.

Ett alternativ till att lista komponenter inne i noder är att använda beroendepilen och stereotypa den till <<deploy>>.



”USECASE DIAGRAM”

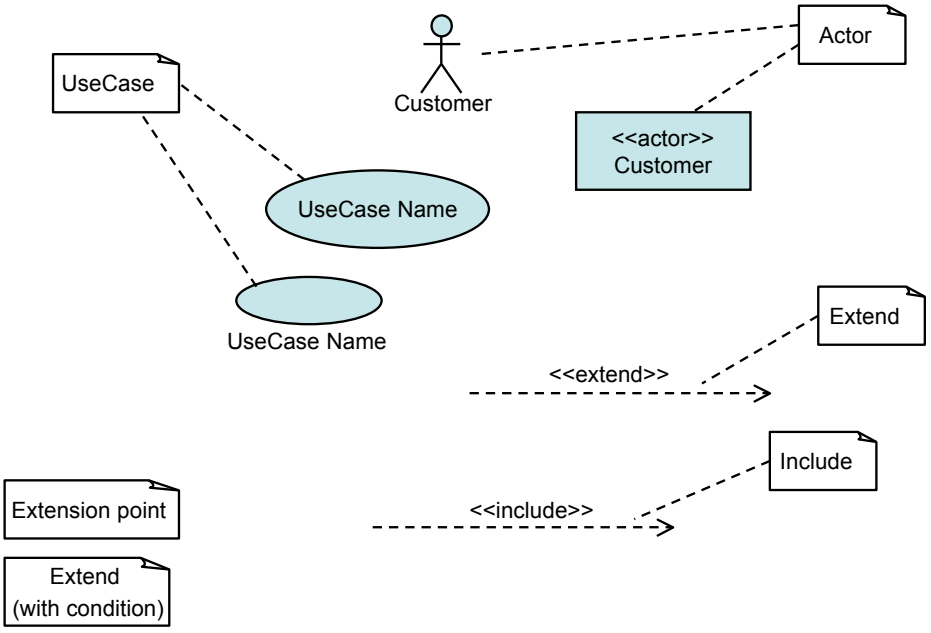
“Use Case Diagram” visar relationen mellan aktörer och systemet samt de användningsfall som aktörerna är involverade i.

- I UML 2.0 finns det tre olika sätt att återanvända användningsfall. Tidigare fanns andra varianter/namn men de som ska användas från och med nu är:
  - <<include>>
  - <<extend>>
  - Arv

”Use Case Diagram”, När...

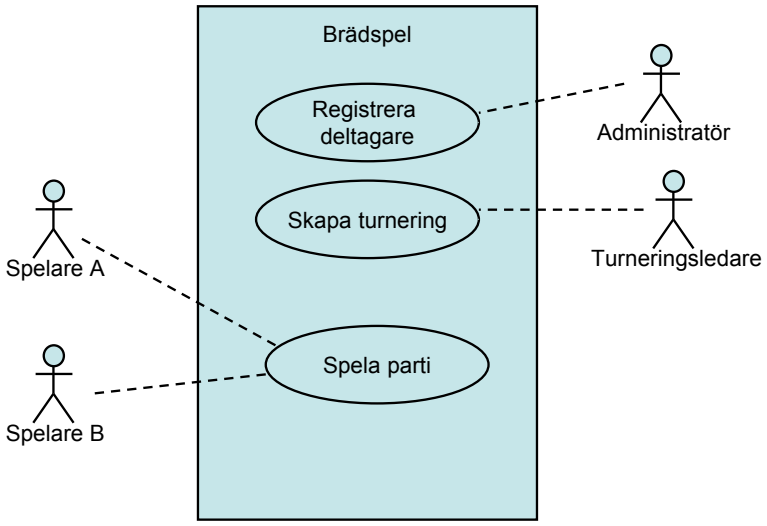
- Användningsfallsdiagram används för att ge en överblick över de användningsfall som finns och deras kopplingar till aktörer.
- Det centrala vid användningsmodellereing är användningsfallspecifikationen. Diagrammet visar endast hur dessa specifikationer och aktörer hänger ihop.
- Användningsfall används för att beskriva kraven på systemen ur ett beställarperspektiv. Det ska inte användas utifrån ett tekniskt perspektiv.

”Use Case Diagram”, UML-element



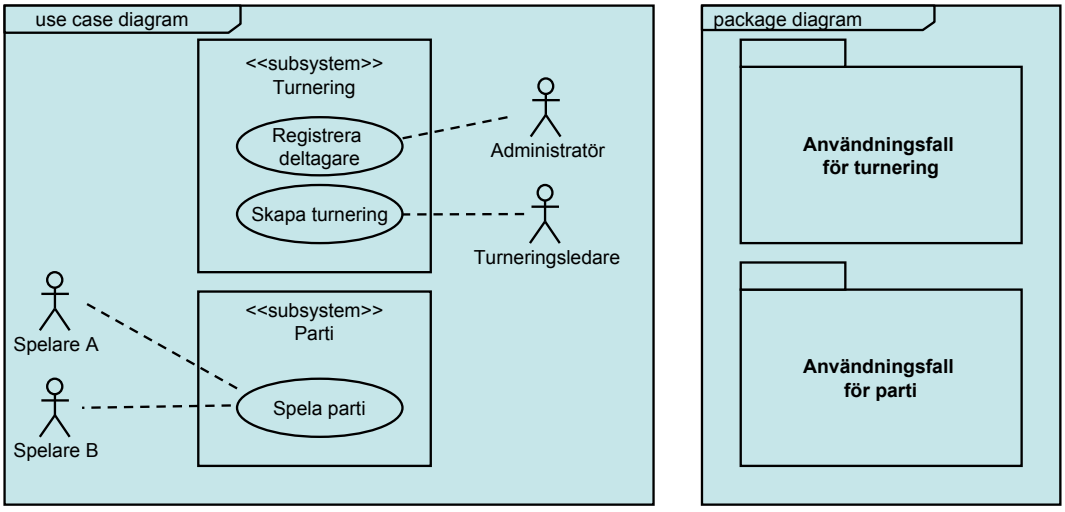
”Use Case Diagram”, Exempel

Det är vanligt att man använder en rektangel för att avgränsa det fysiska systemet från aktörerna.



”Use Case Diagram”, Exempel

Det går att gruppera användningsfall och aktörer i paket. I exemplet delas programmet upp i två delar (subsystem). I paketdiagrammet till höger ingår användningsfallen i respektive paket.



”ACTIVITY DIAGRAM”

Aktivitetsdiagram beskriver en högre nivå av verksamhetsprocesser. Beteenden beskrivs antingen som dataflöden eller som kontrollflöden. Aktivitetsdiagram används även för att modellera komplex logik inom system.

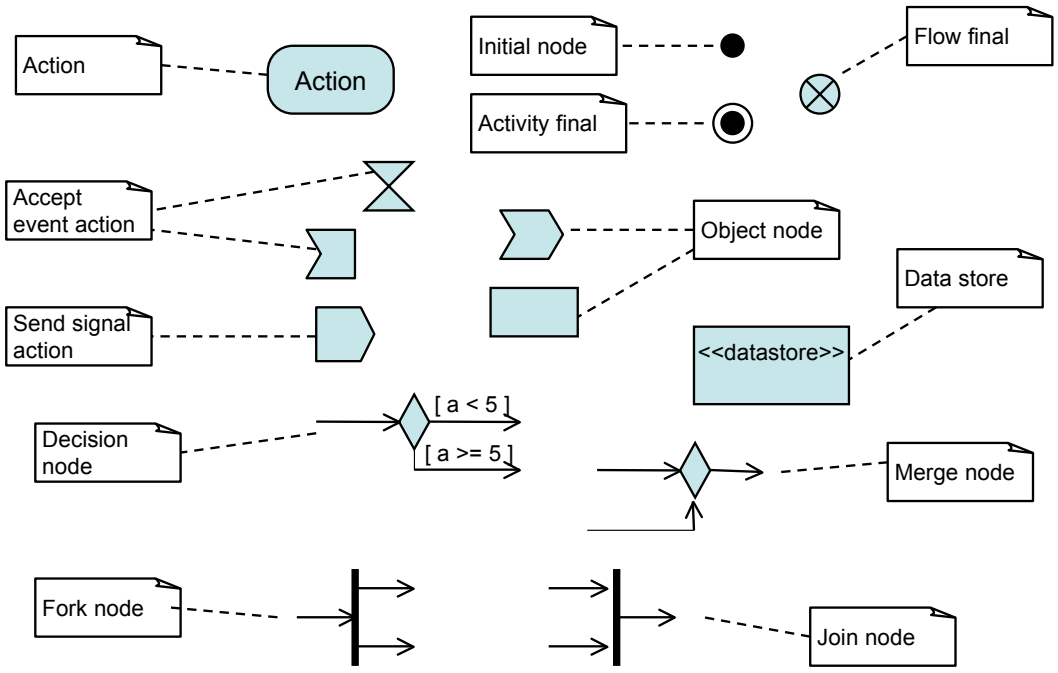
- I tidigare versioner av UML var aktivitetsdiagram en variant av tillståndsdiagram (men där tillstånden var aktiviteter istället).
- I UML 2.0 har diagrammet utökats och blivit ett kraftfullt diagram för att modellera alla typer av aktivitetsflöden.
- En nyhet är att man nu hanterar informations-element i samma diagram som aktiviteter.

”Activity Diagram”, När...

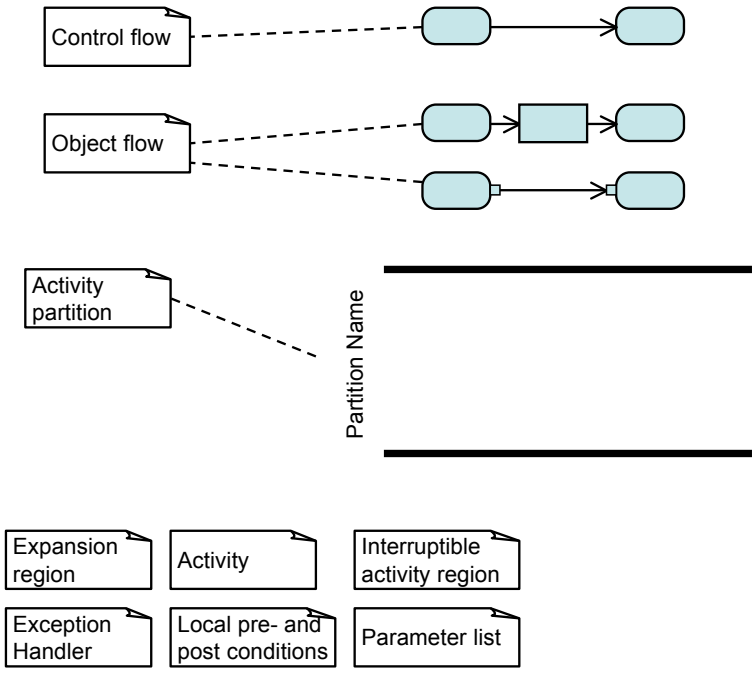
Aktivitetsdiagram används:

- För att beskriva verksamhetsflöden.
- För att beskriva komplex logik.
- För att beskriva flödet i ett användningsfall.
- För att beskriva parallella flöden och processer.
- För att beskriva allmäna processer, exempelvis processer för mjukvaruutveckling.

”Activity Diagram” UML-element

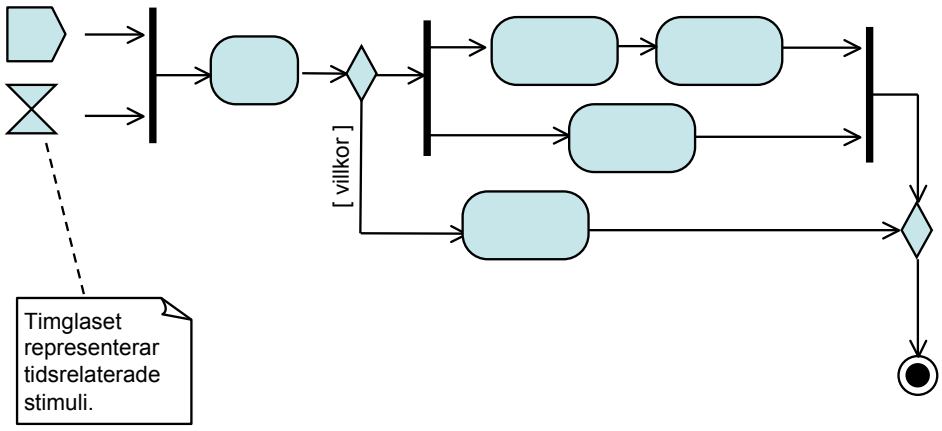


forts...



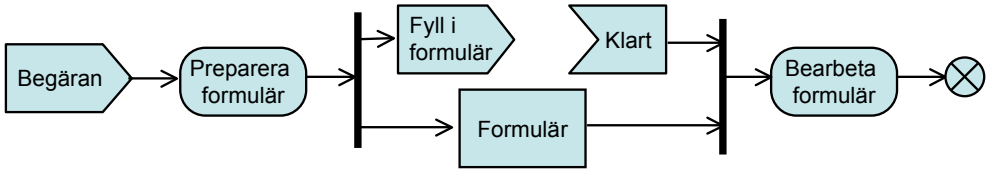
”Activity Diagram”, Exempel

Schematiskt exempel på aktivitetsdiagram:



”Activity Diagram”, Data

Aktivitetdiagram med ett dataelement i form av ett formulär.



”COMMUNICATION DIAGRAM”

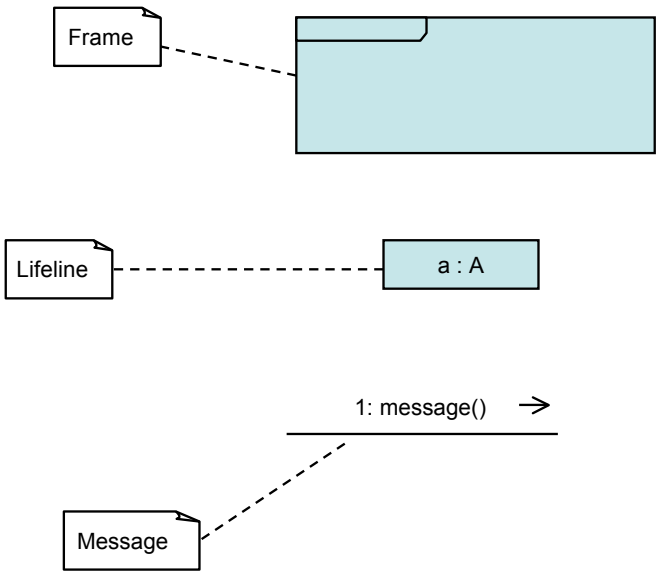
Kommunikationsdiagram används för att visa hur anrop sker mellan objekt (eller mer korrekt ”lifelines”). Det centrala för diagrammet är att visa strukturen och strukturens koppling till hur kommunikation mellan objekt sker. Anropen sekvensnumreras för att visa i vilken ordning anropen sker.

- Namnet ”Communication diagram” är nytt i UML 2.0. Tidigare kallades diagrammet för ”Collaboration diagram” (samarbetsdiagram).
- Sekvensdiagram och kommunikationsdiagram uttrycker motsvarande typ av information, men det visas på olika sätt.

”Communication Diagram”, När...

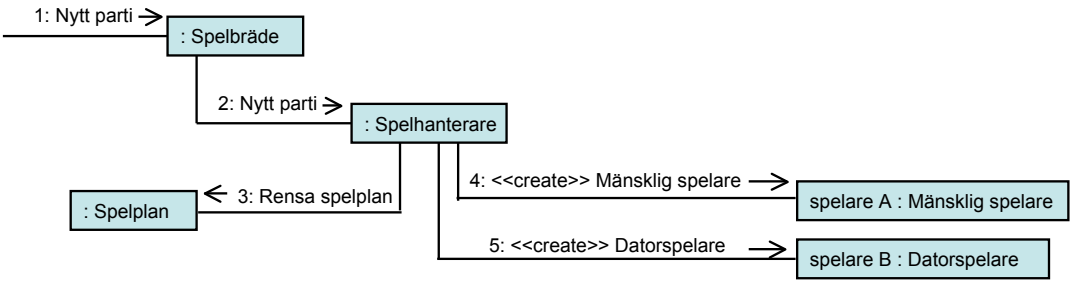
- Diagrammet används för att visa det dynamiska flödet i ett system.
- Historiskt sett är det objekt och meddelande i objektorienterade system som visualiseras, men diagrammet går att använda mer generellt.
- Vid modellering av kod är diagrammet användbart för att skaffa sig en överblick över mekanismer där många objekt är inblandade. Resultatet av en sådan övning kan bli att man strukturerar om indelningen i subsystem eller paket.
- Välj sekvensdiagram när ordningen mellan anrop ska studeras. Välj kommunikationsdiagram när ansvarsfördelning och vägar för kommunikation är viktigt.

”Communication Diagram”, UML-element



”Communication Diagram”, Exempel

Detta diagram visar uppstarten av ett parti. Exemplet är från tidig analysfas.



”SEQUENCE DIAGRAM”

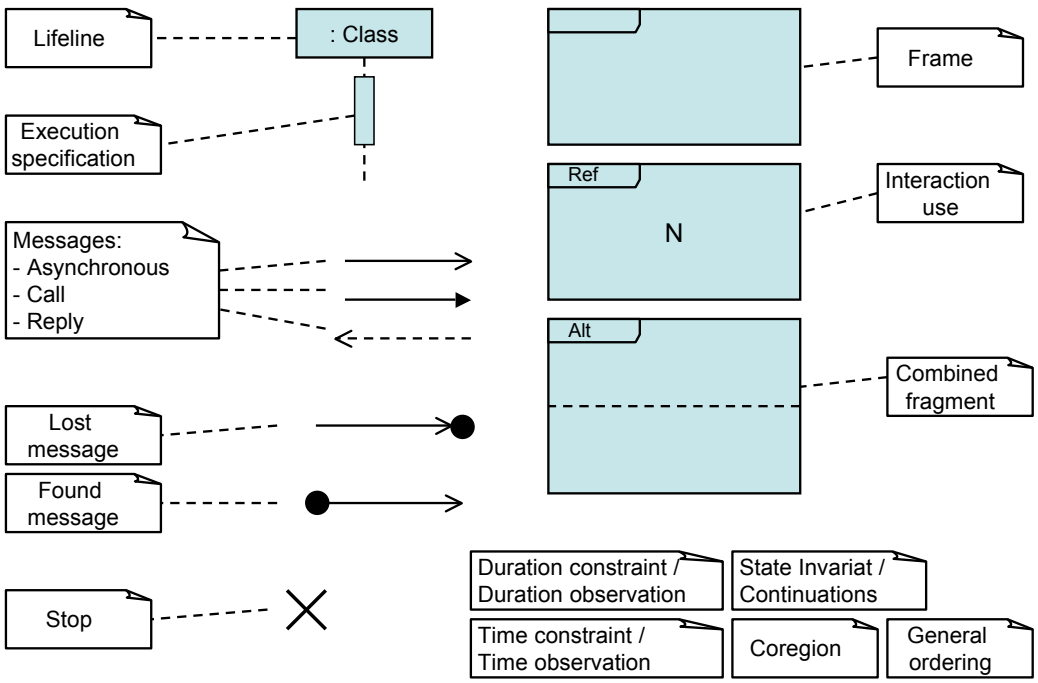
Detta diagram används för att beskriva en sekventiell logik. Ordningen för meddelanden som skickas till objekt (”lifelines”) beskrivs.

- Sekvensdiagram har en hård koppling till hur program exekveras.
- Kommunikationsdiagram beskriver motsvarande information men visualiserar den på ett annat sätt.

”Sequence Diagram”, När...

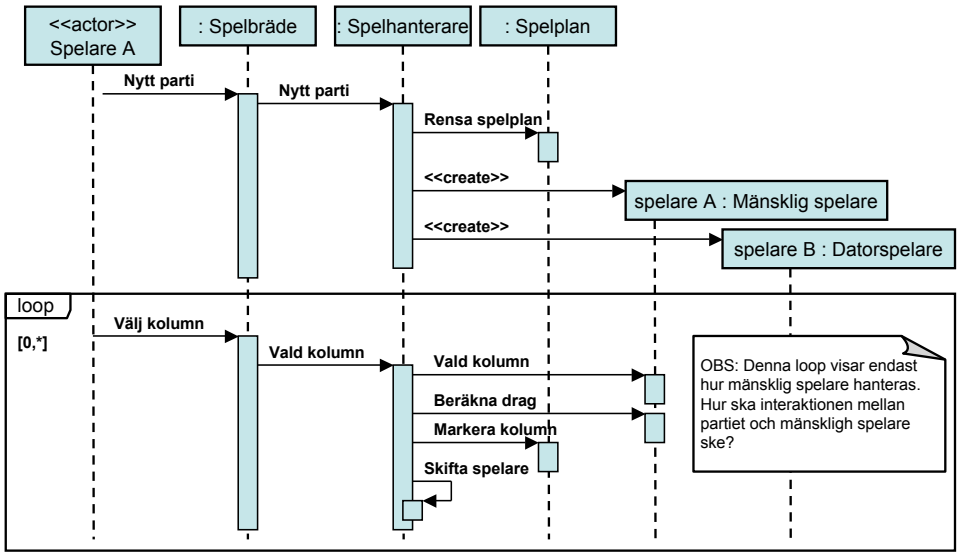
- Ett vanligt användningssätt är att skrivbordstesta sitt system med sekvens-diagram. Det blir mycket tydligt i fall någon klass saknas och man ser vilka metoder som måste implementeras i respektive klass.
- Vid realisering av användningsfall är sekvensdiagram ett av de vikigare verktygen.
- I vissa verktyg kan befintlig kod visualisera i form av sekvensdiagram.

”Sequence Diagram” UML-element



Fyra i rad: Sekvensdiagram nr 1

Detta diagram visar en tidig fundering över hur sekvensen av anrop kan ske. Ännu är inte behovet av en separat speltråd identifierat. Genom att rita motsvarande diagram för två datorspelare kan man få ideér till generell lösning för båda spelartyperna.



”INTERACTION OVERVIEW DIAGRAM”

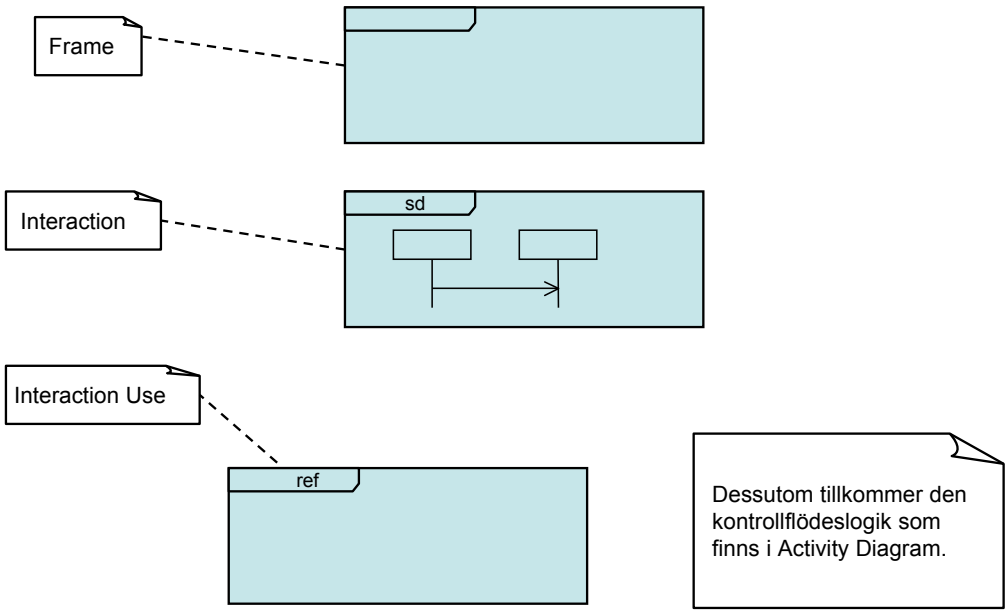
“Interaction Overview Diagram” är en typ av activity diagram som beskriver växelverkan genom ett antal aktivitetsdiagram för att tydliggöra det övergripande kontrollflödet. Noder i diagrammet kan vara interaktions-diagram.

- Diagramtypen är ett nytt tillägg i UML 2.0.
- Grovt sett är diagramtypen ett aktivitetsdiagram där noderna i diagrammet är andra diagram.

”Interaction Overview Diagram”, När...

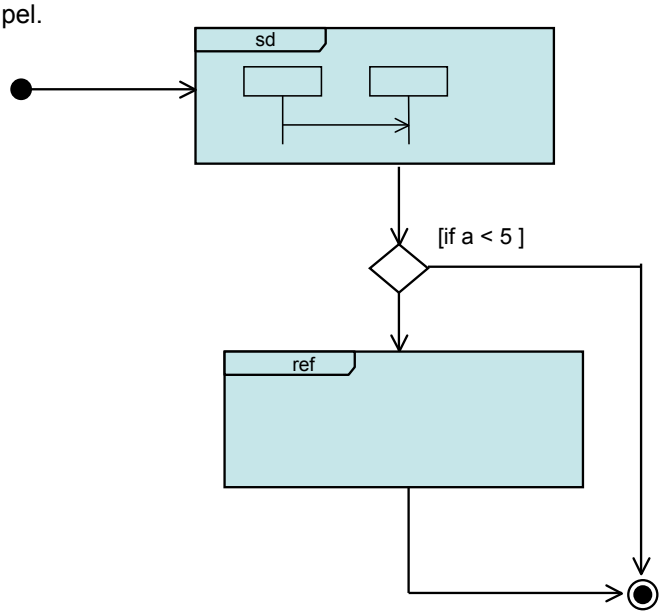
- Vid “top-down”-modellering kan man börja med att beskriva det övergripande flödet med hjälp av “Interaction Overview Diagram”.
- Vid “bottom-up”-modellering när diagrammet lämpligt att ta till när andra interaktionsdiagram inte längre får plats på en sida. Då behövs en uppdelning i flera diagram och detta diagram får vara det som håller diagrammen samman.
- Diagrammet fungerar bra både vid verksamhetsmodellering och för att beskriva programlogik.

”Interaction Overview Diagram”, UML-element



”Interaction Overview Diagram”, Exempel

Schematiskt exempel.



”STATE MACHINE DIAGRAM”

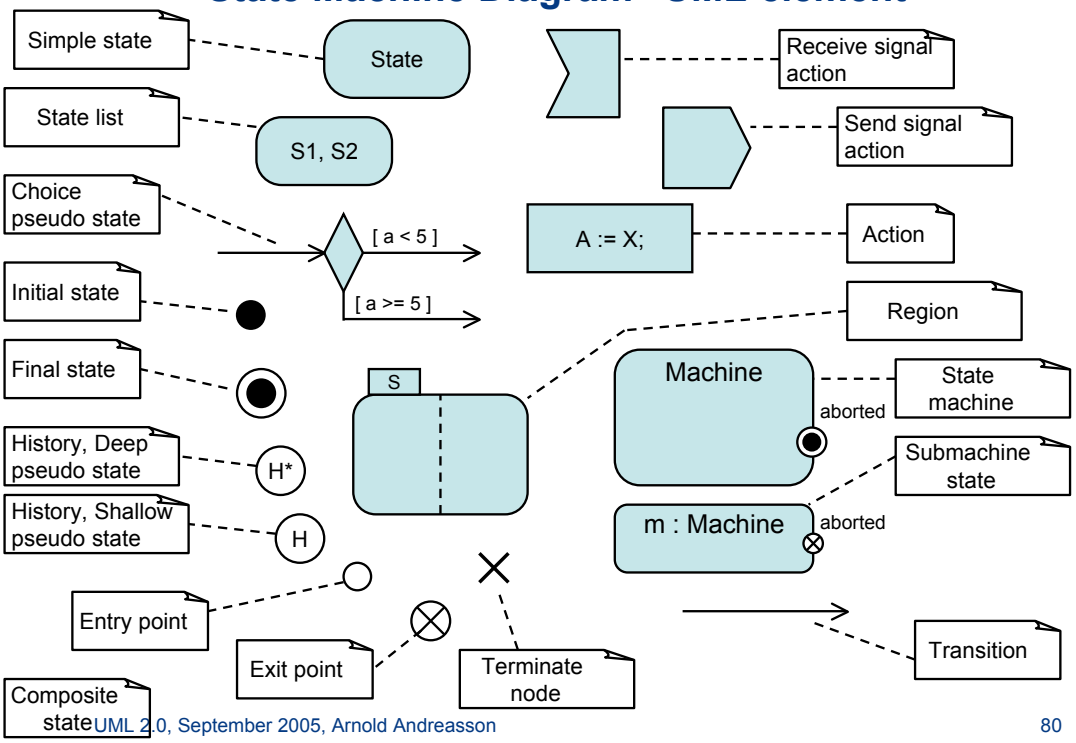
“State Machine Diagram” beskriver de tillstånd ett objekt eller en interaktion kan vara i samt övergångar till andra tillstånd.

- Namnet ”State Machine Diagram” är nytt i UML 2.0. Tidigare benämndes detta diagram State Diagram, State Chart Diagram eller State Transition Diagram.
- Denna diagramtyp har en egen historik utanför UML-världen och har lyfts in som en del i UML.

”State Machine Diagram”, När...

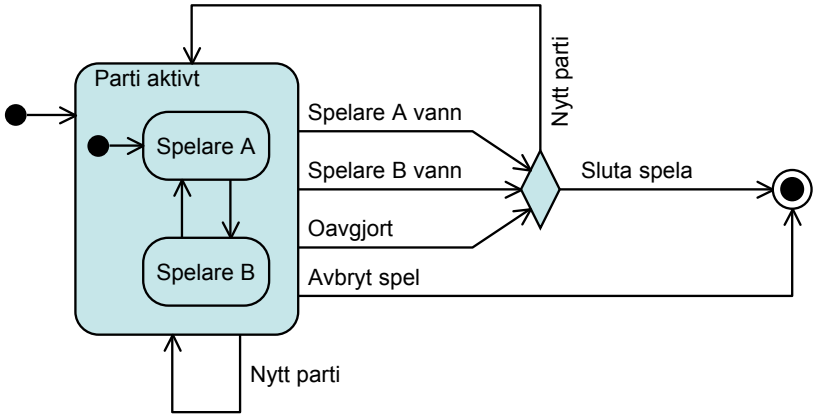
- För att beskriva tillstånd och tillåtna förändringar i tillstånd för en klass eller en grupp av samverkande klasser.
- För att beskriva tillstånd och tillståndsförändringar vid verksamhets-modellering.
- För att beskriva en användardialog med dialogrutor som tillstånd (men det finns olika skolor och åsikter inom detta område).
- För att visualisera en ”finit automat” och andra tillståndsmaskiner.

”State Machine Diagram” UML-element



Fyra i rad: ”State Machine Diagram”

De tillstånd och tillståndsövergångar som spelmotorn i Fyra i rad ska kunna hantera beskrivs i nedanstående diagram.



”TIMING DIAGRAM”

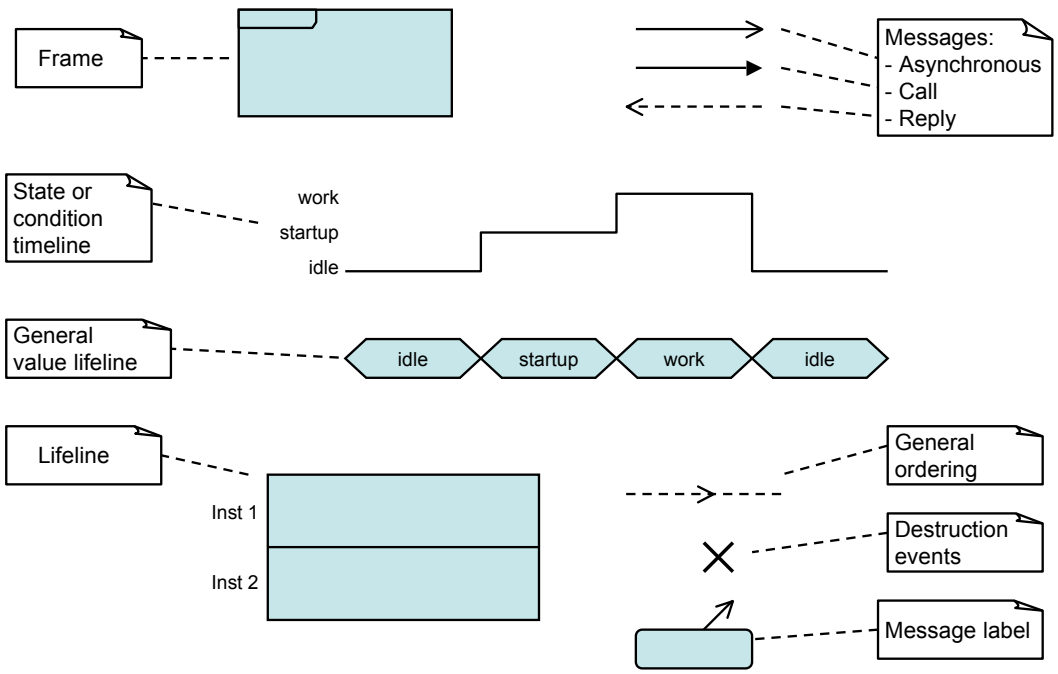
”Timing Diagram” är ett diagram som visar förändringar över en tidsaxel. Diagrammet används för att visa förändringar i tillstånd i relation till olika händelser eller stimuli (exempelvis klockpulser) och när i tiden dessa förändringar/händelser sker i förhållande till varandra.

- ”Timing Diagram” är ett nytt tillägg i UML 2.0, men diagramtypen har använts länge i andra sammanhang.

”Timing Diagram”, När...

- Använd diagramtypen när tidsaspekter är viktigare än sekvensaspekter.
- Hårdvarukonstruktörer har använt diagramtypen länge för att specificera vad som händer i kommunikationsprotokoll och på databussar och här lämpar det sig mycket bra.
- Diagrammet är användbar då man beskriver tidskritiska förlopp vid verksamhetsmodellering.
- Vid systemutveckling kan diagrammet användas för modellering av samspelet och synkronisering av exekveringstrådar.
- Diagrammet har ingen direkt koppling till kod och kan inte direkt användas för kodvisualisering eller kodgenerering.

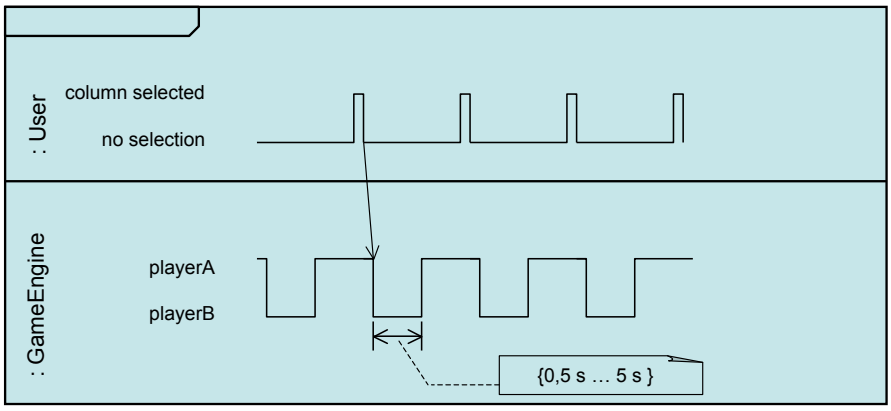
”Timing Diagram”, UML-element



Fyra i rad: ”Timing Diagram”

Detta exempel beskriver synkroniseringbehovet mellan trådar i spelet Fyra i rad.

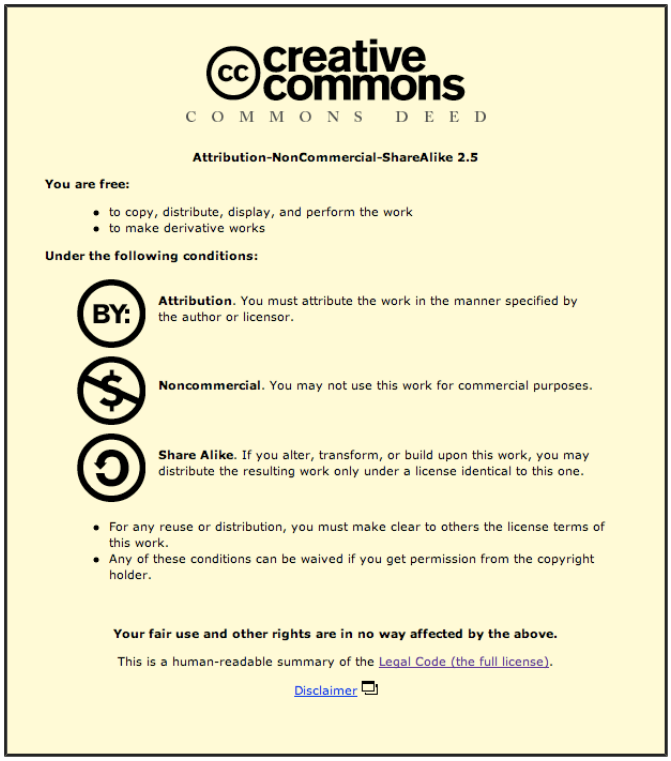
- När ett nytt parti startar skapas en ny tråd i GameEngine. När tråden exekveras växlar GameEngine mellan Player A och Player B.
- Spelare A är en mänsklig spelare (:User) och här behövs synkronisering mellan den tråd som hanterar användargränssnittet och tråden som hanterar pågående parti. Spelare B:s tid styrs av exekveringstid och eventuell delay.



AVSLUTNING, Allmänna råd

Några råd till dig som ska ta fram UML-diagram:

- Varje diagram ska ha ett syfte.
- Varje diagram ska ha en målgrupp (målgrupp kan vara du själv).
- Blanda inte för mycket information i ett och samma diagram. Beskriv bara sådant som är kopplat till syftet.
- Lägg inte tid på att rita diagram som ingen kommer att läsa (= saknar identifierad målgrupp).
- Vissa saker beskrivs bättre och snabbare i kod/metakod eller text än i diagram (beror på målgruppens referensramar).
- Lägg in en notering (“Note”) med projekt, författare, datum och beskrivning/syfte per diagram om diagrammet inte har en tydlig plats i ett dokument.
- Lägg gärna in noteringar för sådant som är oklart i diagrammet.
- De flesta diagram ritas för hand och är temporära. Välj ut de bästa av dessa temporära diagram och dokumentera med rit eller UML-verktyg.
- Lägg ner mycket tid på en ”plansch” för projektet. I projektplanschen är det ofta lämpligt att blanda diagramtyper.



creative commons

COMMONS DEED

Attribution-NonCommercial-ShareAlike 2.5

You are free:

- to copy, distribute, display, and perform the work
- to make derivative works

Under the following conditions:

BY: **Attribution**. You must attribute the work in the manner specified by the author or licensor.

Noncommercial. You may not use this work for commercial purposes.

Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under a license identical to this one.

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

Your fair use and other rights are in no way affected by the above.

This is a human-readable summary of the Legal Code (the full license).

Disclaimer