

# Ramverk för utveckling av J2EE-applikationer



*Decerno har utvecklat ett ramverk för J2EE-applikationer som minimerar nätverkstrafiken mellan skikten. Dessutom ingår rik funktionalitet som väsentligen ökar produktiviteten; t ex validering och SQL-generering.*

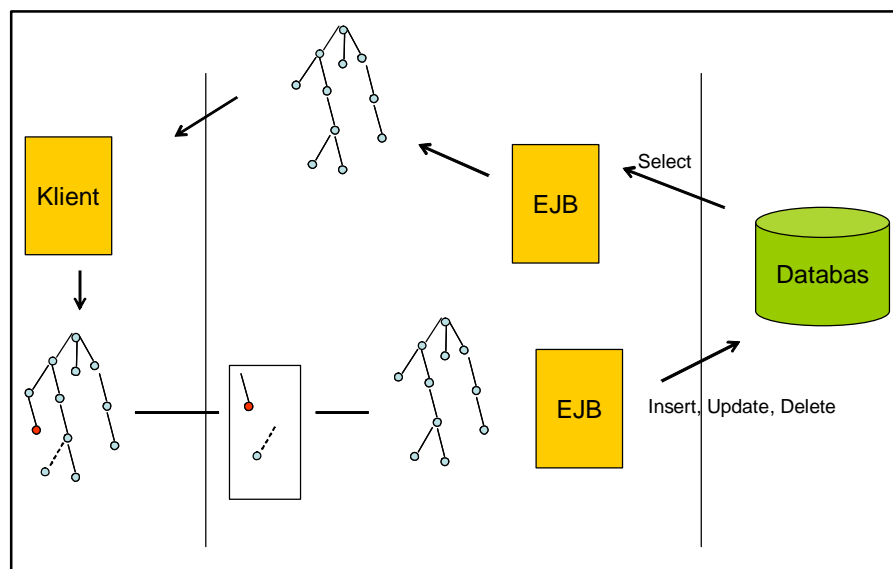
# Ramverk för utveckling av J2EE-applikationer

Dagens flerskiktade arkitektur erbjuder många tekniska fördelar, som t ex skalbarhet och tillförlitlighet. Men de är också svåra att bemästra och många projekt tar betydligt längre tid än beräknat. De många skikten gör applikationen svåröverskådlig och felsökning kan bli en mardröm.

Decerno har utvecklat ett ramverk för J2EE som förbättrar produktiviteten och dessutom löser en rad tekniska problem.

Alla applikationer baserade på J2EE har affärslogiken centralt placerad på en server med Java-böner. Klienten anropar bönorna över nätet för att hämta och spara information. Om värden hämtas ett i taget med get-metoder blir det många anrop och nättrafiken kan bli en flaskhals.

I stället används ett värdeobjekt (Value Object) för att kapsla in en större datamängd som kan överföras i ett anrop. Värdeobjektet skapas av en Java-böna på servern och fylls med aktuella värden från databasen. Väl överförd till klienten kan attributen göras direkt tillgängliga för alla objekt genom att definiera dem som publika. Resultatet är en väsentligt minskad nätverkslast.



*Ramverket JavaSD överför bara den förändrade informationen i värdeobjektet mellan skikten.*

Decernos ramverk JavaSD är i första hand avsett för att på ett strukturerat och effektivt sätt överföra information mellan klient och server. I båda riktningarna. Ramverket överför dessutom bara information som är förändrad. Dels för att minska mängden information som behöver serialiseras. Men också för att ytterligare sänka nätverkstrafiken.

Grovt förenklat är JavaSD en sammansatt datastruktur – värdeobjekt. Till varje värdeobjekt kopplas metainformation som beskriver innehållet. Denna information kan

till exempel utnyttjas för automatisk generering av SQL-satser. Metainformationen ger också stöd för gemensamma kontroller på klient och server.

På data i värdeobjektet är det också möjligt att knyta användarprivilegier ända ned på fältnivå. Vidare kan man ansluta lyssnare som informeras om alla förändringar av data, vilket underlättar utveckling av GUI-applikationer.

Värdeobjekt kan sammanfogas i en trädstruktur bestående av godtyckligt antal nivåer. Strukturen speglar naturligtvis användningen i applikationen. I många fall finns behov av komplexa bilder som t ex stöder relationer av typen order – orderrader. I ett annat fall kan man vilja läsa upp all information om ett ärende.

## JavaSD – en datastruktur

JavaSD håller reda på vilka förändringar som skett i det sammansatta värdeobjektet. Datavärden håller reda på sina originalvärden. Nya och borttagna objekt markeras. De borttagna objekten läggs för sig, etc.

Det ger flera fördelar:

- Klienten kan visa vad som är förändrat
- Om ett värde uppdateras till ursprungsvärdet kommer det att betraktas som oförändrat
- Användaren kan ges möjlighet att ångra förändringar
- Deltamängder av förändrad information kan skapas

När klienten vill spara sina ändringar serialiseras endast den förändrade informationen och skickas till serverns Java-böna. Bönan tar emot informationen och inför ändringarna i sitt eget värdeobjekt. Därefter kontrolleras data ännu en gång, varefter ändringarna införs i databasen.

Värdeobjektet vet alltid själv om det är förändrat. Det är möjligt att på godtycklig nivå i datastrukturen fråga ett objekt om det är förändrat. Det innefattar då även alla objekt på lägre nivå i datastrukturen. När förändringarna är överförda till databasen återställs ”dirty”-markeringen i värdeobjekten på klient och server.

## Värdeobjekten genererar SQL

När det gäller datalagring är entitetsböornas vara eller icke vara ett hett diskussionsämne. Decerno anser att entitetsbönor ger få fördelar men många nackdelar. Speciellt Container Managed Persistence varierar från applikationsserver till applikationsserver. Vid byte av applikationsserver så måste CMP-deskriptorerna skrivas om.

JavaSD är därför inte byggt för att fungera tillsammans med entitetsbönor av typen CMP. Däremot har JavaSD mycket bra stöd för att applikationen själv hanterar datalagringen. Antingen genom stateless sessionsbönor, vilket Decerno rekommenderar, eller Bean Managed Persistence.

I värdeobjekten tillhandahåller programmeraren metadata som används då lagring skall ske i databasen. Metadata är t ex namn på tabeller och kolumner. JavaSD använder sedan informationen för att generera den SQL-kod som skickas till databasen.

Applikationsprogrammeraren behöver därför i allmänhet aldrig själv skriva någon SQL-kod.

SQL-generatorn eliminerar en stor felkälla då de flesta kodningsfel i SQL inte upptäcks förrän koden exekveras. Även i de fall då manuell SQL-kod måste skapas ger JavaSDs klasser ett bra stöd.

Den av JavaSD genererade SQL-koden är databasoberoende och följer SQL-standard. Kommunikationen med databasen sker via JDBC. Möjlighet finns att komplettera genererad SQL-kod eller att skriva den helt själv.

## Mapping objekt till databas

JavaSD består av ett antal paket med basklasser. Det mest centrala paketet hanterar, värden, värdeobjekt och grupper av värdeobjekt (objektset). Ett värdeobjekt kan liknas vid en rad i en databastabell. Värdeobjektets fält motsvaras av kolumner i databasen. Ofta motsvarar ett värdeobjekt en rad, eller del av en rad, i en databastabell. Men det kan mycket väl innehålla information från flera tabeller. Värdeobjekten i ett objektset motsvarar då ett antal rader i en databastabell eller vy.

JavaSD innehåller en mängd klasser (värdetyper) för olika sorters värden. Exempel på värdetyper är *SdDoInteger*, *SdDoString* och *SdDoDate*. Varje applikation kan skapa egna värdetyper utgående från dem som finns i JavaSD. Värdetyperna kan innehålla max- och min-värden, domäner, kontrollmetoder mm.

Domäner kan vara hårdkodade för datatypen eller utgöras av data från databasen. Domändata kan cachas på klienten. De godkända värdena kan till exempel kopplas till en kombobox i användargränssnittet.

Kontrollmetoder för värdetyper exekveras då ett värde förändras. Om ett fel upptäcks förkastas värdet och en exception kastas.

Fält som motsvarar värdeobjektets nycklar definieras inte direkt i värdeobjektet. Istället skapas en primärnyckelklass innehållande dessa fält. Primärnyckeln läggs sedan in som ett primärnyckelfält i värdeobjektet.

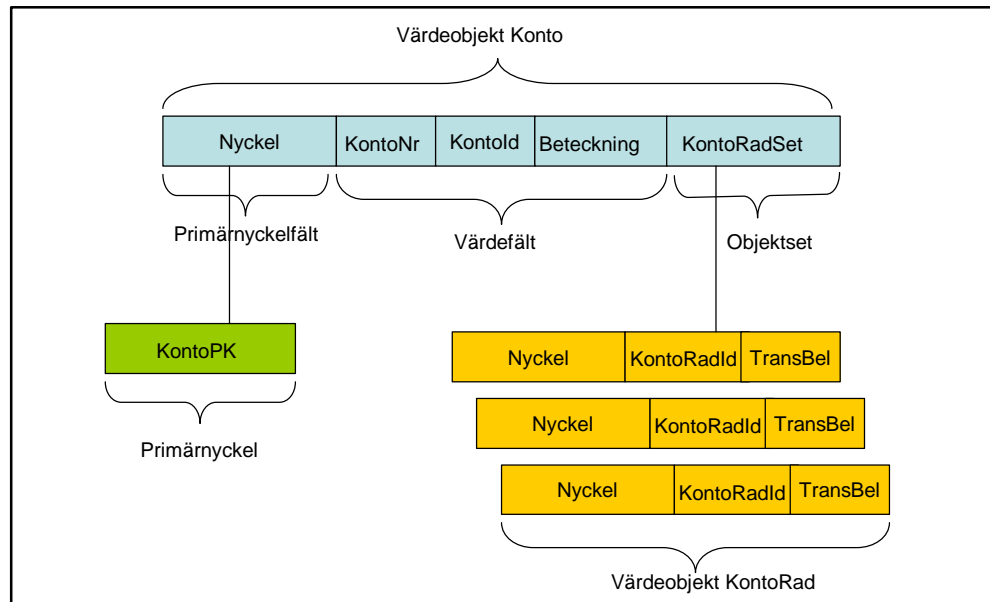
Innan samtliga fältvärden i en nyckel är definierade finns ett unikt ID i primärnyckeln. Det möjliggör unik identifikation av värdeobjektet då det skickas mellan klient och server, även i det fall då servern är ansvarig för skapandet av nyckelvärdet.

Funktionen är praktisk exempelvis när insert görs på en post med automatgenererad nyckel. Nyckeln finns ännu inte när klienten skapar posten och skickar över den till servern.

JavaSD primärnyckelklasser kan användas som primärnycklar i EJB:s Entity Beans.

*Exempel: Värdeobjektet Konto, som motsvarar ett konto, har en primärnyckel, kallad "Nyckel", av klassen KontoPK. Det innehåller dessutom en mängd andra värdefält, såsom KontoNr och KontoId av klassen SdDoLong och Beteckning av klassen SdDoString.*

Ett objektset kan innehålla ett antal värdeobjekt. Därför kan ett objektset liknas vid en tabell eller vid i databasen, även om det inte behöver finnas en ren mappning.



JavaSD tillåter uppbyggnad av komplexa värdeobjekt i form av trädstrukturer.

Ett värdeobjekt kan innehålla fält i form av inbäddade objektset. På så sätt kan man beskriva master-detail förhållanden.

*Exempel: Värdeobjektet Konto innehåller det inbäddade objektsetet KontoRadSet som i sin tur innehåller kontots kontorader i form av värdeobjekt från klassen KontoRad.*

Ett värdeobjekt kan även innehålla fält i form av andra inbäddade värdeobjekt. Det kan användas när man har en ett till ett-mappning mellan objekten. Objekten bör ha gemensamma nycklar och det inbäddade objektets nycklar låses mot det överordnade objektets nycklar.

Genom att objektset kan innehålla objekt - och objekt i sin tur kan innehålla inbäddade objektset - är det möjligt att skapa datastrukturer med trädform. Datastrukturerna kan innehålla komplexa delmängder från databasen.

Datastrukturerna är serialiserbara, så att de kan överföras mellan klient och server. JavaSD möjliggör byggande av mycket komplexa trädstrukturer. Det är dock inte ett självändamål att bygga stora strukturer. Behoven varierar beroende på applikation.

## Metainformation och arv

I värdeobjekten kopplas metainformation till de olika fälten. Här kan man lägga information om längder, max- och min-värden, fältrubriker, kontroller mm.

Värdeobjekt kan tillföras nya fält genom arv. All metainformation kommer att ärvas automatiskt.

Mycket av behörigheten i en J2EE-applikation sköts normalt på metodnivå. Men viss behörighet kan behöva styras på fältnivå. Varje fält i en JavaSD-struktur kan få behörighets-information i form av attributen "läsbar" och "uppdateringsbar". För objektset kan man också beskriva om hela värdeobjekt får läggas till och tas bort.

Behörighetsinformationen kan även utnyttjas på klienten för att inaktivera komponenter som representerar icke uppdateringsbara värden. Till exempel kan ett fält "gråas" ut.

Olika typer av lyssnare kan anslutas till en datastruktur. Lyssnarna får information om förändrade datavärden, nya objekt, borttagna objekt mm. För en swing-klient innebär det till exempel att man bygger modeller för att ansluta swing-komponenter till JavaSDs datastrukturer. Färdiga swing-modeller finns till exempel för trädstrukturer och tabeller.

Lyssnare kan även anslutas till ett värdefält. Det kan framförallt utnyttjas av enkla swing-komponenter på klienten (textfält, komboboxar, kryssrutor etc.). Komponentens målas om när lyssnaren indikerar att värdet har ändrats.

JavaSD är anpassat för användargränssnitt byggda med Java-swing. Men ramverket kan också styras om mot andra grafiska gränssnitt.

## Sökvillkor och kontroller

JavaSD innehåller även klasser för att förenkla för programmeraren att skapa komplexa sökbilder. Ofta är det besvärligt att hålla reda på alla parenteser och "AND/OR" för att bilda söksträngen dynamiskt.

Sökvillkoret byggs istället upp i form av en trädstruktur helt oberoende av lagringssätt på servern. Klienten kopplar värden till löven i trädstrukturen beroende på vad användaren matat in. Klasserna på serversidan skapar sedan utifrån strukturen samt tabell- och kolumnnamn ett korrekt SQL-villkor. Nästan alla tänkbara sökkriterier är möjliga; t ex =, like, in, > och <.

Att kontrollera data är en väsentlig del i all programmering. Ramverket stöder kontroller, både på klient och på server. På klienten för att ge snabbare återkoppling. Och på servern för att den inte kan lita på klienten!

Upptäckta fel identifieras alltid med det objekt där kontrollen gjordes. Man vet därför alltid var felet uppstod. Det gör det möjligt för klienten att positionera till den komponent som hanterar det felaktiga objektet. Användaren kan i sin tur uppmärksammas genom att markera fältet eller visa ett meddelande.

Fel- och andra meddelanden kan presenteras för användaren på lokalt språk. Klasser i JavaSD hanterar resursfiler med de olika texterna.

Innan klienten försöker spara ett objekt eller objektset ska den alltid anropa metoden *check* som kan göra en fullständig kontroll av hela den underliggande strukturen. Metoden anropar i sin tur funktionen *checkItem*, som kan ersättas av egen funktionalitet:

```
protected void checkItem() throws SdIllegalValueException {
    if(itemCount() == 13) {
        throw new SdIllegalValueException(this,
            "Antal personer får inte vara 13 stycken");
    }
}
```

Fältkontroller i objekt görs genom att man i descriptorn definierar en anonym inre klass som implementerar interfacet *ISdDoFieldCheck*. Detta interface definierar endast metoden *checkField(ISdDoValue)*:

```

c_oPostNummerDescriptor.setFieldCheck(new ISdDoFieldCheck() {
    public void checkField(ISdDoValue value) throws SdIllegalValueException {
        ISdDoInteger postNummer = (ISdDoInteger)value;
        if(postNummer.isNormal()) {
            int nPostNummer = postNummer.getInt();
            if(nPostNummer < 10000 || nPostNummer > 99999) {
                throw new SdIllegalValueException(value, "Postnummer måste ha 5 siffror", null);
            }
        }
    }
});

```

Kontroller kan också sättas deklarativt direkt på attributnivå. Exempelvis *KontoNr.setMaybeNull(false)*;

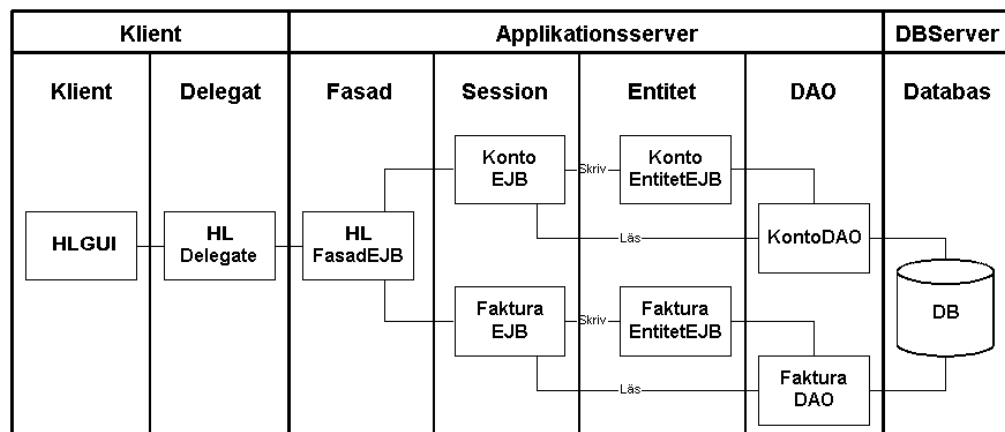
Kontroller av min- och max-värden för fält kan sättas via fältdescriptorn. För strängfält kan man definiera min- och max-längder. För andra fält kan man skriva egna metoder för *setMinLength()* respektive *setMaxLength()*.

Metoden *getMinimumLength()* i *SdDoString* returnerar den största minlängd som satts via värdetypen och fältdescriptorn. På motsvarande sätt returnerar *getMaximumLength()* den minsta maxlängd som satts via värdetypen och fältdescriptorn. Det är dessa värden som används vid kontroll av värdets längd.

## Lyckade projekt

JavaSD dikterar inte hur datastrukturerna ska hanteras på klient och server. Det är varje projekts ensak att definiera hur strukturerna ska byggas upp.

JavaSD har använts i projekt på Riksgäldskontoret. Applikationen hanterar alla betalningar, krediter och anslag för internbankens kunder, Sveriges myndigheter och statliga verk. Totalt hanterar systemet 500 miljarder kr per år och har 600 användare.



Arkitekturen på Riksgälden bygger på JavaSD och rymmer över 2 000 java-klasser.

Projektet har använt sin egen modell med uppdelning i olika skikt i form av delegat på klienten och fasad-, session- och entitets-objekt på servern.

Systemet utnyttjar också referenser mellan bönor på servern vilket innebär att datastrukturer inte behöver serialiseras mellan bönorna. Det förbättrar prestanda, samtidigt som bland annat felhanteringen blir enklare.

## *Decerno*

*Decerno bygger pålitliga system – ett löfte som innebär flera saker. Vi bygger stabila verktyg som skänker trygghet i arbetet. Samtidigt framtidssäkra system, som är lätta att anpassa när verksamheten förändras.*

*Pålitlighet innebär också säker leverans på utsatt tid och inom givna kostnadsramar. För att skapa pålitliga system krävs ett långsiktigt tänkande. Både teknik och metoder måste stödja kvalitet i form av bl a testbarhet, enkelt underhåll och tillförlitlighet. Decerno har därför ett långt drivet kvalitetsarbete.*

*Under de tjugo år Decerno varit verksamma har vi drivit över hundra projekt till ett lyckat slut.*