(12) **PATENT**

(19) NO (11) **331543** (13) **B1**

NORGE

(51) Int Cl.
G06F 17/00 (2006.01)
G06F 9/00 (2006.01)
G06F 9/06 (2006.01)
G06F 9/44 (2006.01)
G06F 9/45 (2006.01)
G06F 9/46 (2006.01)
G06F 13/00 (2006.01)
G06F 15/16 (2006.01)
G06F 12/00 (2006.01)
G06T 3/00 (2006.01)

Patentstyret

(21)	Søknadsnr	20041263	(86)	Int.inng.dag og søknadsnr	
(22)	Inng.dag	2004.03.25	(85)	Videreføringsdag	
(24)	Løpedag	2004.03.25	(30)	Prioritet	2003.03.26, US, 401244
(41)	Alm.tilgj	2004.09.27			
(45)	Meddelt	2012.01.23			

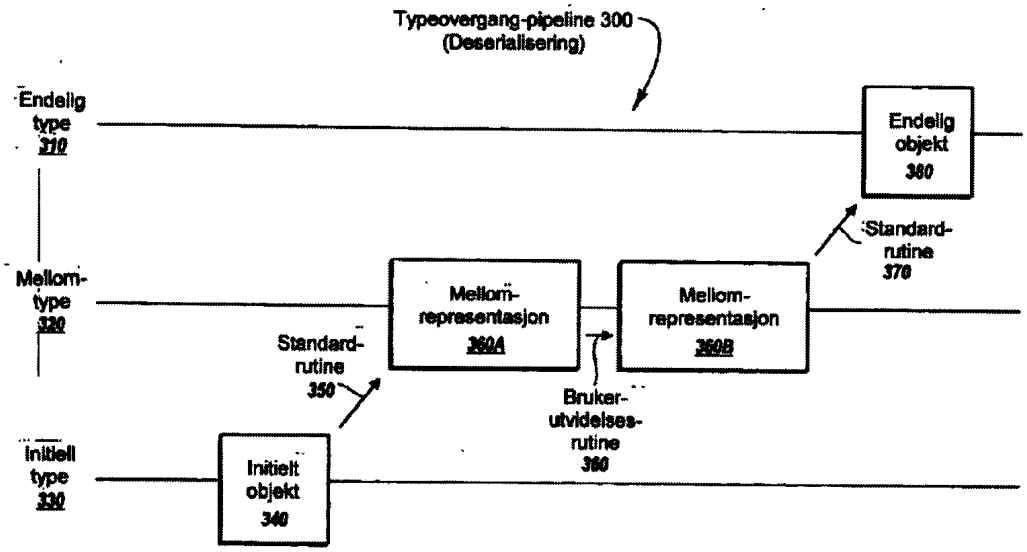
(73) Innehaver Microsoft Corp, One Microsoft Way, US-WA98052-6399 REDMOND, USA
(72) Oppfinner Stefan H Pharies, 525 E Roy Street, Apartment 301, US-WA98102 SEATTLE, USA
Sowmy K Srinivasan, 9755 228th Terrace NE, US-WA98053 REDMOND, USA
Natasha H Jethanandani, 800 137th Avenue NE, C9-204, US-WA98005 BELLEVUE, USA
Yann Erik Christensen, 817 16th Avenue, US-WA98122 BELLEVUE, USA
Elena A Kharitidi, 413 239th Avenue NE, US-WA98074 SAMMAMISH, USA
Douglas M Purdy, 28634 NE 63rd Way, US-WA98014 CARNATION, USA
(74) Fullmektig Bryn Aarflot AS, Postboks 449 Sentrum, 0104 OSLO, Norge

(54) Benevnelse **Typeoverganger.**

(56) Anførte publikasjoner EP 1 030 253 A1

(57) Sammendrag

Fremgangsmåter, systemer og dataprogramprodukter for å konvertere et objekt av én type til et objekt av en annen type som muliggjør endring eller individuell tilpasning av kjøremåten til konverteringsprosessen. Konverteringen kan bli utført i en utvidbar serialiseringsmotor som serialiserer, deserialiserer og konverterer objekter av forskjellig type. Kjøremåten til serialiseringsmotoren blir endret av én eller flere utvidelsesrutiner som implementerer de ønskede spesialtilpasninger eller utvidelser uten å kreve utskiftning av andre eksisterende rutiner. Basert på typeinformasjon, som er identifisert for et initielt objekt, blir objektet konvertert til en mellomrepresentasjon som muliggjør modifikasjon under kjøring, inklusive modifikasjon av objektnavn, objekttyper og objektdata. Mellomrepresentasjonen av det initielle objektet blir modifisert i henhold til utvidelsesrutiner som endrer kjøremåten til serialiseringsmotoren, og mellomrepresentasjonen blir konvertert til et endelig objekt og en endelig type.



Foreliggende oppfinnelse vedrører serialisering av objekter. Mer spesifikt vedrører foreliggende oppfinnelse fremgangsmåter, systemer og dataprogramprodukter for å konvertere objekter av én type til objekter av en annen type ved hjelp av utvidelsesrutiner som endrer kjøremåten til en serialiseringsmotor uten at det er nødvendig å skifte ut andre eksisterende rutiner i serialiseringsmotoren.

I generell forstand refererer serialisering til det å konvertere enkeltstående eller innrettinger av (nestede) minnelagrede objekter til en lineær sekvens av bit-oktetter som er egnet for overføring til en fjernlokasjon, persistent lagring på disk, etc. Motsatt tar deserialisering den lineære sekvensen av bit-oktetter og oppretter de motsvarende enkeltstående eller innrettinger av minnelagrede objekter. Utført etter hverandre resulterer typisk serialisering og deserialisering i at det blir generert et eksakt klon av det opprinnelige objektet.

Tradisjonelt har serialiseringskode blitt skrevet som et monolittisk program, uten mulighet for individuell tilpasning bortsett fra å skifte ut hele programmet. Mangelen på mulighet for individuell tilpasning eller utvidbarhet gjør serialiseringsmekanismen lite fleksibel for brukere, inklusive utviklere og andre interesserte parter. Med en monolittisk implementasjon er stegvise forbedringer eller individuelle tilpasninger for å adressere et gitt problem direkte ofte ikke mulig, og kan kreve tungvinte utenomgåelser eller ganske enkelt utelukke visse ønskede operasjoner. Dersom individuelle tilpasninger skal foretas uansett, er typisk standardrutiner som implementerer nyttig funksjonalitet ikke tilgjengelige for utvikleren, og må derfor bli implementert på nytt, noe som i vesentlig grad øker (og ofte uoverkommeliggjør) arbeidet som er nødvendig for å utvikle den ønskede tilpasningen. Som følge av dette kan typisk ny funksjonalitet bli lagt til i serialiseringskoden av utviklerne av serialiseringskoden, noe som hindrer sluttbrukere i å utvikle sine egne forbedringer og forbedre eksisterende funksjonalitet.

Selv om en eksakt kopi av et objekt ofte er målet for serialisering og deserialisering, kan konvertering av objekttyper, -navn og -data under kjøring være ønsket i noen tilfeller. Som angitt ovenfor kan serialisering og deserialisering for eksempel bli anvendt for å overføre et objekt til en fjernlokasjon. Fjernlokasjonen kan forvente visse typer objekter, objektdata og objektnavn som er forskjellige fra

kildeobjektet. Tradisjonell serialiseringskode kan være skrevet for å utføre objekt-transformasjoner, men transformeringen kan ikke bli lagt til under kjøring og er den samme for alle brukere, noe som ikke tar hensyn til at forskjellige brukere vil kunne ha forskjellige behov. Selv om en gitt transform kan være veldig viktig for en gitt bruker på et gitt tidspunkt, kan transformens generelle relevans være ubetydelig for brukermassen som helhet, og derfor ikke implementert.

Tradisjonell serialiseringskode har også en tendens til å tilby liten fleksibilitet med hensyn til det å identifisere objekter å konvertere, eller å basere konverteringer på data inneholdt i et objekt. Følgelig er det ønskelig med fremgangsmåter, systemer og dataprogramprodukter for å konvertere objekter av én type til objekter av en annen type, basert på spesialiserte rutiner for å endre serialisering og deserialisering under kjøring, uten at det er nødvendig å re-implementere standardrutiner.

I et første aspekt tilveiebringer oppfinnelsen en fremgangsmåte, som utføres i et datasystem som omfatter en utvidbar serialiseringsmotor som serialiserer og deserialiserer dataobjekter av forskjellig type, for å konvertere et initielt objekt av en initiell type til et endelig objekt av en endelig type, der fremgangsmåten muliggjør endring av kjøremåten til serialiseringsmotoren gjennom én eller flere utvidelsesrutiner uten at det er nødvendig å bytte ut én eller flere eksisterende rutiner i serialiseringsmotoren, idet fremgangsmåten omfatter trinn for å identifisere typeinformasjon for et initielt objekt av en initiell type som mottas for kjøretidsprosessering av serialiseringsmotoren, basert på typeinformasjonen, konvertere det initielle objektet til en mellomrepresentasjon av det initielle objektet som er egnet for kjøretidsmodifisering, modifisere mellomrepresentasjonen av det initielle objektet i henhold til én eller flere utvidelsesrutiner, og med det endre kjøremåten til serialiseringsmotoren, konvertere mellomrepresentasjonen av det initielle objektet til et endelig objekt av en endelig type, og å utsette modifisering av mellomrepresentasjonen til mellomrepresentasjonen konverteres til det endelige objektet for å unngå å måtte bufre en modifisering av mellomrepresentasjonen.

I et andre aspekt tilveiebringer oppfinnelsen et dataprogramprodukt som implementerer en utvidbar serialiseringsmotor for å konvertere ett eller flere initielle objekter av én eller flere initielle typer til ett eller flere endelige objekter av én eller

flere endelige typer, der kjøremåten til serialiseringsmotoren kan bli endret uten at det er nødvendig å re-implementere eksisterende deler av serialiseringsmotoren, idet dataprogramproduktet omfatter ett eller flere datamaskinlesbare medier som bærer datamaskin-eksekverbare instruksjoner i form av programmoduler, der programmodulene omfatter: en refleksjonsmodul som kan bli byttet ut under kjøring, for å identifisere typeinformasjon for et initielt objekt av en initiell type som er mottatt for prosessering av serialiseringsmotoren, én eller flere konverteringsmoduler som kan bli byttet ut under kjøring, for å generere og modifisere en mellomrepresentasjon av det initielle objektet basert på den identifiserte typeinformasjonen, idet den ene eller de flere kjøretid-utskiftbare konverteringsmodulene omfatter én eller flere utvidelsesrutiner som endrer kjøremåten til serialiseringsmotoren, og en genereringsmodul som kan bli byttet ut ved kjøretid, for å opprette et endelig objekt av en endelig type basert på mellomrepresentasjonen generert av konverteringsmodulen, hvor den ene eller de flere kjøretid-utskiftbare konverteringsmodulene er i stand til å utsette én eller flere endringer av mellomrepresentasjonen av det initielle objektet inntil mellomrepresentasjonen blir konvertert til det endelige objektet for å unngå bufferkrav i forbindelse med det å utføre den ene eller de flere modifiseringene av mellomrepresentasjonen.

Det beskrives fremgangsmåter, systemer og dataprogramprodukter for å konvertere et objekt av en initiell type til et objekt av en endelig type, og muliggjør endring eller individuell tilpasning av kjøremåten til konverteringsprosessen. I henhold til eksempler på utførelse av foreliggende oppfinnelse som er beskrevet mer utførlig nedenfor, kan en utvidbar serialiseringsmotor serialisere, deserialisere og konvertere objekter av forskjellig type. Kjøremåten til serialiseringsmotoren endres av én eller flere utvidelsesrutiner som implementerer de ønskede individuelle tilpasninger eller utvidelser. Disse utvidelsesrutinene endrer kjøremåten til serialiseringsmotoren uten å kreve utskiftning av andre eksisterende rutiner.

I ett utførelseseksempel blir typeinformasjon identifisert for et initielt objekt mottatt av serialiseringsmotoren for prosessering. Basert på typeinformasjonen blir det initielle objektet konvertert til en mellomrepresentasjon som muliggjør modifisering under kjøring, inklusive endring av objektnavn, objekttyper og objektdata. Mellomrepresentasjonen av det initielle objektet modifiseres i henhold til én eller flere

utvidelsesrutiner som endrer kjøremåten til serialiseringsmotoren, og mellomrepresentasjonen blir konvertert til et endelig objekt av en endelig type.

Mellomrepresentasjonen av det initielle objektet kan omfatte et objektnavn, en objekttype og objektdata, som alle kan bli modifisert av utvidelsesrutinene. Mellomrepresentasjonen kan også bli modifisert av én eller flere standardrutiner i serialiseringsmotoren. Modifikasjonen av mellomrepresentasjonen kan være basert på en gitt struktur eller modell innenfor typeinformasjonen, objektdata i det initielle objektet, metadata eller kombinasjoner av foregående.

Når det initielle objektet er et minnelagret objekt, serialiserer serialiseringsmotoren det initielle objektet for å generere det endelige objektet. Det endelige objektet kan bli formatert i XML (eXtensible Markup Language) eller i et annet format som er egnet for å representere et serialisert objekt. Tilsvarende, når det endelige objektet er et minnelagret objekt, deserialiserer serialiseringsmotoren det initielle objektet for å generere det endelige objektet. Det endelige objektet kan bli instansiert og initialisert som del av deserialiseringsprosessen. I noen tilfeller kan både det initielle objektet og det endelige objektet være minnelagrede objekter, eller begge kan være serialiserte objekter, for eksempel når serialiseringsmotoren utfører en objektkonvertering. For å redusere bufferkravene kan modifiseringen av mellomrepresentasjonen utsettes til mellomrepresentasjonen blir konvertert til det endelige objektet.

Ytterligere særtrekk og fordeler ved oppfinnelsen vil bli vist i beskrivelsen som følger, og vil delvis være åpenbare fra beskrivelsen eller kan læres gjennom praktisering av oppfinnelsen. Særtrekkene og fordelene ved oppfinnelsen kan realiseres og oppnås ved hjelp av de redskaper eller virkemidler og kombinasjoner som spesifikt er utpekt i patentkravene. Disse og andre særtrekk ved foreliggende oppfinnelse vil tydeliggjøres bedre av den følgende beskrivelsen og kravene, eller kan læres gjennom praktisering av oppfinnelsen som vist i det følgende.

For å beskrive hvordan de ovenfor angitte og andre fordeler og særtrekk ved oppfinnelsen kan oppnås, vil en mer detaljert beskrivelse av oppfinnelsen som kort beskrevet ovenfor bli gitt med henvisning til spesifikke utførelsesformer av denne som er illustrert i de vedlagte figurene. Idet man forstår at disse figurene kun viser typiske

utførelsesformer av oppfinnelsen, og derfor ikke skal anses som begrensende for dens ramme, vil oppfinnelsen bli beskrevet og forklart mer spesifikt og detaljert ved hjelp av de vedlagte figurene, der:

Figur 1 illustrerer et eksempel på serialiseringsmodul og serialiseringsinfrastruktur ifølge foreliggende oppfinnelse,

Figurene 2-4 viser konvertering av objekter i forbindelse med eksempler på "pipeliner" eller behandlingskøer for serialisering, deserialisering og typekonvertering.

Figurene 5A-5B viser eksempler på handlinger og trinn i fremgangsmåter for serialisering, deserialisering og konvertering av objekter i henhold til foreliggende oppfinnelse, og

Figur 6 illustrerer et eksempel på system som utgjør et egnet driftsmiljø for foreliggende oppfinnelse.

Foreliggende oppfinnelse omfatter fremgangsmåter, systemer og dataprogramprodukter for å konvertere objekter av en initiell type til objekter av en endelig type, og muliggjør endring eller individuell tilpasning av kjøremåten til konverteringsprosessen. Utførelsesformer av foreliggende oppfinnelse kan omfatte én eller flere spesialinnrettede og/eller én eller flere generelle datamaskiner som omfatter forskjellig maskinvare, som diskutert mer detaljert nedenfor i forbindelse med figur 6.

Figur 1 illustrerer et eksempel på serialiseringsmodul og serialiseringsinfrastruktur 100 (også kjent som en serialiseringsmotor) ifølge foreliggende oppfinnelse. For en objektinstans 110 genererer serialiseringsmodulen 100 et motsvarende XML-objekt 150. Tilsvarende, for et XML-objekt 160, genererer serialiseringsmodulen 100 en motsvarende, deserialisert objektinstans 170. Det skal bemerkes at i denne søknaden, betegnelsen "serialisering" ofte blir anvendt som en generisk betegnelse som omfatter serialisering (f.eks. det å konvertere enkeltstående eller innrettinger av minnelagrede objekter til en lineær sekvens av bit-oktetter som er egnet for overføring til en fjernlokasjon, persistent lagring på disk, etc.), deserialisering (det å opprette, fra den lineære sekvensen av bit-oktetter, de motsvarende enkeltstående eller innrettinger av minnelagrede objekter), konvertering (det å konvertere ett objekt til et annet), etc. Dette er for eksempel tilfelle her når

serialiseringsmodulen 100 serialiserer, deserialiserer og konverterer objekter av forskjellig type.

Serialiseringsmodulen 100 omfatter én eller flere refleksjonsmoduler 120, én eller flere konverteringsmoduler 130 og én eller flere genereringsmoduler 140. I dette utførelseseksempelet konverterer serialiseringsmodulen 100 en mottatt minnelagret objektinstans 110 til et XML-objekt 150 som er egnet for overføring til en fjernlokasjon, og konverterer en mottatt XML-objektinstans 160 til en minnelagret objektinstans 170. Naturligvis er minnelagrede objekter og XML-objekter kun eksempler på objekttyper som kan bli opprettet eller mottatt av serialiseringsmodulen 100. Hver av modulene i serialiseringsmodulen 100 (refleksjonsmodulene 120, konverteringsmodulene 130 og genereringsmodulene 140) kan bli byttet ut under kjøring for brukerdefinert serialisering, deserialisering eller konvertering.

Refleksjonsmodulene 120 har ansvar for å identifisere typeinformasjon for mottatte objektinstanser 110 og mottatte XML-objekter 160. Typeinformasjonen kan omfatte lagrede eller mottatte metadata som er assosiert med administrerte typer i et miljø med administrert kode. Alternativt kan typeinformasjonen bli levert til refleksjonsmodulene 120 fra ulike kilder, omfattende automatisert generering under kompilering, manuell generering, standard typeinformasjon, etc.

Konverteringsmodulene 130 konverterer mellom objekter av forskjellig type. Eksempler på konverteringsprosesser er beskrevet mer i detalj nedenfor i forbindelse med figurene 2-4. Konverteringen mellom forskjellige objekter kan være vilkårlig kompleks og omfatte generering av mellomobjekter. Deler av denne kompleksiteten kan omfatte det å konvertere basert på data i et objekt og typestrukturer assosiert med et objekt. For eksempel kan hvilke konverteringer som blir utført avhenge av visse objekttyper eller typenavn, eksistens av gitte navnede eller typede egenskaper på en type, eksistens av en egenskap med gitte metadata tilknyttet, objektnavn assosiert med et objekt, etc. Konverteringen kan bli utsatt til generering av det endelige objektet for å redusere eller unngå krav til bufring som ellers kan være en del av det å konvertere ett objekt til et annet.

Genereringsmodulene 140 har ansvar for å generere det endelige objektet som frembringes av serialiseringsmodulen 100. I tilfellet med et XML-objekt 150

oppretter genereringsmodulen 140 objektet – den genererer den relevante XML-koden for objektet – og kan skrive objektet til en datastrøm. I tilfellet med en objektinstans 170 vil genereringsmodulen 140 instansiere og initialisere eller bygge opp objektet.

Som nevnt ovenfor er serialiseringsmodulen 100 også kjent som en serialiseringsmotor. Som vist i figur 1, utgjøres serialiseringsmotoren av et antall ordnede sett av moduler. Sammen har disse modulene ansvar for alle operasjoner. En individuell modul er kjent som en typeovergang, fordi, som beskrevet mer i detalj nedenfor, moduler konverterer fra én type til en annen (eller danner overganger mellom forskjellige typer). En typeovergang gjør det mulig å konvertere typer og instanser og/eller å holde rede på informasjon om et objekt som blir serialisert, deserialisert eller konvertert. Med henvisning til figurene 2-4, er et ordnet sett av typeoverganger kjent som en typeovergang-pipeline, og svarer generelt til et ordnet sett av konverteringsmoduler 130. For hver av operasjonene som blir utført av serialiseringsmotoren kan det eksistere en separat typeovergang-pipeline. Det finnes pipeliner for serialisering (f.eks. figur 2), deserialisering (f.eks. figur 3), konvertering (f.eks. figur 4), objektkopiering, etc. Informasjon som gjelder generelt for alle tre figurene er presentert nedenfor, før en individuell diskusjon av hver av figurene 2-4.

For de eksempelvise pipelinene som er vist i figurene 2-4 er koden (én eller flere moduler) som har ansvar for serialisering, deserialisering og konvertering av objekter implementert i form av et antall forhåndsdefinerte typeoverganger. Disse modulene blir plassert i den aktuelle pipelinen og anvendt under kjøring. (De stiplede kvadratene i figur 1 er ment å representere tilgjengelige typeovergangsmoduler for anvendelse i forskjellige typeovergang-pipeliner.) En stor del av det tilgjengelige API-et for det eksempelet på serialiseringsmotor som er vist i figur 1 er kun en innpakning av dette forhåndsdefinerte settet av pipeliner. Dette demonstrerer serialiserings-motorens utvidbarhet – serialiseringsmotoren er bare et sett av abstrakte pipeliner. Den faktiske implementasjonen av spesifikk logikk er tilveiebrakt i pluggbare moduler som kan skiftes ut når som helst.

I de eksemplene på typeovergang-pipeliner som er illustrert i figurene 2-4, er en gitt typeovergang i stand til å konvertere én av tre typer objekter: et initielt objekt,

et mellomobjekt og et endelig objekt. I figur 4 er det initielle objektet et administrert kode-objekt og det endelige objektet er et XML-objekt basert på W3C (World Wide Web Consortium) sin Infoset-standard. Mellomobjektet eller mellomrepresentasjonen vist i alle tre figurene er en konstruksjon som eksisterer internt i serialiseringsmotoren, og, som beskrevet mer i detalj nedenfor, representerer et utvidelsespunkt. Mellomrepresentasjonen er et foranderlig objekt som er basert på en foranderlig type. Som sådan tjener den foranderlige typen til å definere oppførsel og lagring av typede data, og det foranderlige objektet tjener til å lagre typede data og manipulere de typede dataene gjennom oppførsel definert på typen.

Figur 2 viser et eksempel på typeovergang 200 for å serialisere et minnelagret initielt objekt 240 av initiell type eller format 210. (Som anvendt i beskrivelsen og kravene, skal betegnelsen "type" tolkes i bred forstand til å omfatte hvilke som helst objekttyper eller -formater.) Med bruk av en standardrutine 250 blir det initielle objektet 240 konvertert til en mellomrepresentasjon 260A av en mellomtype eller et mellomformat 220. Som vil bli beskrevet mer i detalj nedenfor, er denne mellom-representasjonen foranderlig, og tillater endring av både objekttyper og objektdata. Allikevel kan mellomformatet 220 og det initielle formatet 210 også være det samme, nært beslektet, litt forskjellig, helt forskjellig, etc.

En brukerdefinert utvidelsesrutine 260 konverterer mellomrepresentasjonen 260A av det initielle objektet 240 til en mellomrepresentasjon 260B. Denne konverteringen kan omfatte endring av objekttyper, objektnavn, objektdata og liknende. Den brukerdefinerte utvidelsesrutinen 260 representerer en kjøreutvidelse av serialiseringsmotoren generelt, og typeovergang-pipelinen 200 spesielt. Merk at anvendelsen av den brukerdefinerte utvidelsesrutinen 260 ikke krevde re-implementering av standardrutinen 250, noe som typisk er tilfelle med konvensjonelle serialiseringsprogrammer.

Standardrutinen 270 konverterer mellomrepresentasjonen 260B til et endelig objekt 280 av endelig type eller format 230. Det endelige objektet 280 er egnet for overføring til en fjernlokasjon, persistent lagring, etc. Følgelig omfatter det endelige formatet 230 til det endelige objektet 280 et bredt spekter av objekttyper. Her, som i andre deler av beskrivelsen, er objekttype, -format og -representasjon generelle

betegnelser som omfatter objektets type og format samt typer, formater, navn og data som kan være inneholdt i et objekt.

Figur 3 viser et eksempel på typeovergang 300 for å deserialisere et objekt 340 av initiell type eller format 330. Tilsvarende som i figur 2 over, konverterer en standardrutine 350 det initielle objektet 340 til en mellomrepresentasjon 360A med en mellomtype eller et mellomformat 320. En brukerdefinert utvidelsesrutine 360 konverterer mellomrepresentasjonen 360A til en mellomrepresentasjon 360B. Merk at mellomtypen 320 representerer én eller flere mellomtyper. Følgelig kan mellom-representasjonen 360A og mellomrepresentasjonen 360B være forskjellige typer, men er likevel korrekt angitt som en mellomtype, i forhold til initiell type 330 og endelig type eller format 310.

En standardrutine 370 konverterer mellomrepresentasjonen 360B til et endelig objekt 380 av en endelig type 310. Ettersom typeovergang-pipelinen 300 utfører deserialisering, er det endelige objektet 380 et minnelagret objekt som er instansiert og populert. Som vil bli beskrevet mer i detalj nedenfor, er typeovergang-pipelinen 300 tilsluttet kode for å instansiere og initialisere objektinstanser. Denne koden kan bli referert til som en instansfabrikk eller skriverfabrikk, og svarer generelt til genereringsmodulene 140 vist i figur 1.

Figur 4 viser et eksempel på typeovergang-pipeline 400 for å konvertere et initielt objekt 440 til et endelig objekt 480. De individuelle typeovergangene vist i figur 4 er i stand til å konvertere ett av tre forskjellige objektyper eller -formater: en administrert kode / CLR-formaterte objekter 410, mellomobjekter / Flex-formaterte objekter 420 og Infoset / XML-formaterte objekter 430. CLR står for "Common Language Runtime" og er en del av Microsofts .NET® administrerte kjøremiljø. Blant annet omfatter fordelene ved CLR integrasjon på tvers av programmeringsspråk, unntaksbehandling på tvers av programmeringsspråk og liknende. Kompilatorer mater ut metadata for å beskrive typer, medlemmer og referanser. Metadataene lagres med koden i den flyttbare CLR-kjørefilen. CLR er naturligvis kun ett eksempel på en type administrert kode. Som antydet i figur 4 kan begge objektene være minnelagrede objekter (f.eks. CLR-formaterte objekter), eller alternativt kan begge

objektene være serialiserte objekter (f.eks. Infoset-formaterte objekter). Med andre ord kan det initielle objektet og det endelige objektet være av samme type.

CLR-objekter 410 er instanser eller forekomster av CLR-typer som inneholder en kombinasjon av data og oppførsel eller funksjonalitet, selv om kun dataene er relevante for serialiseringsformål. Som angitt over er et Infoset-objekt eller en Infoset-represesentasjon 430 formatert i henhold til en W3C-standard for en trestruktur bestående av et predefinert sett av datanoder med gitt semantikk. Et flex-objekt 420 er en konstruksjon som finnes internt i serialiseringsmotoren, og representerer et utvidelsespunkt for den som serialiserer.

Et flex-objekt er et foranderlig objekt som er basert på en foranderlig type. Den foranderlige typen er kjent som en flex-type. I det eksempelet på typeovergang-pipeline 400 som er vist i figur 4, tjener en flex-type samme funksjon som dens motsvarende CLR-type, nemlig å definere oppførsel og lagring av typede data. Tilsvarende tjener et flex-objekt samme funksjon som CLR-objekt, nemlig å lagre typede data og manipulere disse dataene gjennom oppførsel definert på typen. Grunnen til at flex-typer og flex-objekter blir anvendt er at CLR-typer er uforanderlige.

I det eksempelet på typeovergang-pipeline som er vist i figur 4, er det lagt visse begrensninger på typene som kan bli serialisert for å fremme enkelthet og utvidbarhet. Disse begrensningene reduserer antallet forskjellige modeller og permutasjoner som serialisereren trenger å gjenkjenne for å serialisere og deserialisere en gitt type. For dette formål vet serialiseringsmotorer kun hvordan de skal serialisere CLR-objekter hvis type følger noe som er kjent som en kjernemodell. Typer som følger kjernemodellen må enten eksponere sine data som egenskaper (eller felter) eller implementere et gitt funksjonsgrensesnitt (som definerer eksplisitte lese- og skrive-metoder). I tillegg må disse typene implementere en "public" default-instansieringsfunksjon. Typer som ikke følger kjernemodellen kan ikke serialiseres.

Flex-typer og flex-objekter anvendes for å endre formen (medlemmer, funksjonsgrensenitt, etc.) til et gitt CLR-objekt slik at det følger kjernemodellen. For det gitte CLR-objektet kan det bli konstruert en flex-type som eksponerer et forskjellig sett av medlems- og typeinformasjon enn instansens CLR-type. Et flex-objekt som er basert på flex-typen kan bli instansiert som delegerer visse kall til CLR-objektet selv.

Flex-objektet kan også utføre eventuelle konverteringer av dataene i CLR-objektet, enten før eller etter delegering. Som følge av dette kan data i CLR-objektet bli eksponert på ulike måter, inklusive en som følger kjernemodellen. Følgelig kan en typeovergang starte med en objekttype som ikke følger kjernemodellen og generere en objekttype som gjør det.

En typeovergang kan konvertere CLR-objekter, flex-objekter og Infoset-representasjoner på ulike måter. En hvilken som helst gitt typeovergang tar en inntype som den opererer på og en uttype som den produserer eller genererer. Denne uttypen blir sendt til den neste typeovergangen i pipelinen. For den eksemplifiserte typeovergang-pipelinen 400 er følgende konverteringer støttet:

Inntype	Uttype	Beskrivelse
CLR	CLR	Konverterer et CLR-objekt til et nytt CLR-objekt
CLR	Flex	Konverterer et CLR-objekt til et flex-objekt
CLR	Infoset	Konverterer et CLR-objekt til et Infoset-objekt
Flex	Flex	Konverterer et flex-objekt til et nytt flex-objekt
Flex	CLR	Konverterer et flex-objekt til et CLR-objekt
Flex	Infoset	Konverterer et flex-objekt til et Infoset-objekt
Infoset	Infoset	Konverterer et Infoset-objekt til en ny Infoset
Infoset	Flex	Konverterer et Infoset-objekt til et flex-objekt
Infoset	CLR	Konverterer et Infoset-objekt til et CLR-objekt

De forskjellige klassifiseringene av typeoverganger er frembragt for å tilveiebringe serialiseringsmotorens grunnfunksjonalitet. (Selv om figurene 2 og 3 refererer til generiske typer, er henvisninger til disse figurene gjort nedenfor med de spesifikke typene som er illustrert i figur 4 for å gi en bredere sammenheng.)

1. Serialisering konverterer et CLR-objekt til et Infoset-objekt eller en Infoset-representasjon. For å utføre denne operasjonen er det tilveiebrakt en typeovergang-pipeline (så som den vist i figur 2) som omfatter en CLR-til-flex typeovergang (f.eks. standardrutinen 250), et hvilket som helst antall flex-til-flex overganger samt en flex-til-Infoset typeovergang (f.eks. standardrutinen 270).

2. Deserialisering konverterer en Infoset-representasjon til et CLR-objekt. For å utføre denne operasjonen er det tilveiebrakt en typeovergang-pipeline (så som den vist i figur 3) som omfatter en Infoset-til-flex typeovergang (f.eks. standard-rutinen 350), et hvilket som helst antall flex-til-flex overganger og en flex-til-CLR typeovergang (f.eks. standardrutinen 370).
3. Objektkopiering blir anvendt for å generere en dypkopi av et CLR-objekt. For å utføre denne operasjonen er det tilveiebrakt en typeovergang-pipeline som omfatter en CLR-til-CLR typeovergang.
4. Objektkonvertering (figur 4) genererer en dypkopi (det endelige objektet 480) av CLR-objektet eller -objektene (det initielle objektet 440) samtidig med utførelse av eventuelle konverteringer (standard- eller brukerutvidelsesrutinen 460) av instansdataene (mellomrepresentasjonene 460A og 460B). For å utføre denne operasjonen er det tilveiebrakt en typeovergang-pipeline som omfatter en CLR-til-flex typeovergang (standard- eller brukerutvidelsesrutinen 450), eventuelt én eller flere flex-til-flex typeoverganger (standard- eller brukerutvidelsesrutinen 460) som utfører konverteringene, samt en flex-til-CLR typeovergang (standard- eller brukerutvidelsesrutinen 470).
5. Infoset-konverteringen skaper en kopi av og konverterer eventuelt et Infoset-objekt. Tilsvarende som for objektkopieringen er det for å utføre denne operasjonen tilveiebrakt en typeovergang-pipeline som omfatter en Infoset-til-Infoset typeovergang.

De siste tre alternativene er verdt å merke seg på grunn av måten de er implementert. Mens andre implementasjoner bufrer objekt- eller Infoset-data, kan utførelsesformer av foreliggende oppfinnelse utsette konverteringene for å unngå eller redusere bufringskravene. Som følge av dette kan ytelsen og ressursstyringen forbedres betraktelig.

For å støtte de ovennevnte operasjonene tilveiebringer serialiseringsmotoren stamme- eller base-typeoverganger som utfører de aktuelle konverteringene. I figur 4 kan hvilke som helst av standard- eller brukerutvidelsesrutinene 450, 460 og 470 være base-typeoverganger eller brukerdefinerte erstatninger. Ved anvendelse av en utvidbar konfigureringsmekanisme blir de relevante typeoverganger identifisert og

lastet inn i pipeliner under kjøring. Serialiseringsmotoren anvender disse base-pipelinene for å utføre den forespurte operasjonen. Base-typeovergangene kan imidlertid bli byttet ut når som helst, ettersom motoren jobber med abstrakte typeoverganger heller enn spesifikke implementasjoner. I ett utførelseseksempel omfatter en pipeline bare en liste av typeoverganger for pipelinen, slik at endring av listen endrer pipelinen. I dette utførelseseksempelet, når en gitt typeovergang blir anropt for et objekt, anropes ingen andre typeoverganger for dette objektet.

Merk at i ett utførelseseksempel, CLR 410, flex 420 og Infoset 430 svarer til det initielle formatet 210, mellomformatet 220 og det endelige formatet 230 for serialisering som vist i figur 2, og svarer til den endelige typen 310, mellomtypen 320 og den initielle typen 330 for deserialisering som vist i figur 3. Flex-objektet er mellomformatet mellom CLR og Infoset. I dette utførelseseksempelet tillates ikke en typeovergang å konvertere direkte fra CLR til Infoset, eller omvendt. Blant annet bidrar dette til å forenkle den eksemplifiserte serialiseringsmotoren. Mens den grunnleggende funksjonaliteten eller virkemåten til serialiseringsmotoren er definert av base-typeoverganger, finnes det også mange ytterligere trekk (så som støtte for eksisterende programmeringsmodeller) som utviklere kan forvente. Base-typeovergangene kunne ha blitt designet for å implementere disse trekkene, men istedet er det tilveiebrakt et antall grunnleggende flex-til-flex typeoverganger som tjener dette formålet. Denne løsningen sikrer at base-typeoverganger er enkle og utvidbare. Som følge av dette kan utviklere modifisere standardfunksjonalitet og tilveiebringe ny funksjonalitet på egen hånd.

For dette utførelseseksempelet, betrakt en serialiseringsprosess for en CLR-type navnet Person som har to egenskaper: ForNavn og EtterNavn. For å serialisere (se figur 2) en instans av denne typen, er det nødvendig med en pipeline med grunnleggende CLR-til-flex og flex-til-Infoset typeoverganger. Serialiseringsmotoren sender Person-instansen til CLR-til-flex typeovergangen. Denne typeovergangen returnerer en ny instans av et flex-objekt som er basert på og delegerer til Person-instansen. Flex-objektet blir deretter sendt til flex-til-Infoset typeovergangen.

Flex-til-Infoset typeovergangen har ansvar for å transformere eller konvertere flex-objektet til en Infoset-representasjon. Før konverteringen bestemmer den

grunnleggende flex-til-Infoset typeovergangen hvordan den skal avbilde flex-objektets struktur til Infoset. Base-implementasjonen i dette eksempelet anvender et skjemaspråk og definerer avbildninger med midlene definert i dette språket. Siden typeoverganger er utbyttelige, kan det bli introdusert en ny avbildningsmekanisme som omfatter støtte for et nytt skjemaspråk, hvilket representerer et annet utvidelsespunkt i serialiseringsmotoren. Når avbildningsprosessen er avsluttet, konverteres objektet til en Infoset-representasjon som blir skrevet til en strøm.

Som såvidt nevnt ovenfor, er typeovergangene i serialiseringsmotoren tilknyttet skriverfabrikker. Skriverfabrikker har ansvar for opprettelse av en ressurs som er i stand til å skrive data. Selv om ressursen vil kunne skrive data til et hvilket som helst mål, er de mest vanlige destinasjonene datastrømmer (etter serialisering for transport) og CLR-objekter (etter deserialisering) Base-skriverfabrikken for dette utførelseseksempelet returnerer en ressurs som skriver til en brukerlevert datastrøm. Ressursen som genereres av denne fabrikken kan skrive til datastrømmen på et hvilket som helst format den måtte ønske. Som sådan er den ikke låst til XML-serialiseringsformatet, noe som gjør skriverfabrikken utbyttelig og introduserer nok et annet utvidelsespunkt i serialiseringsmotoren.

Deserialisering (se for eksempel figur 3) av Infoset-representasjonen i dette utførelseseksempelet involverer en pipeline som omfatter de grunnleggende Infoset-til-flex og flex-til-CLR typeovergangene. Serialiseringsmotoren sender en brukerlevert datastrøm som representerer Infoset-kilden så vel som hvilken CLR-type (her Person) som blir deserialisert til den første typeovergangen (Infoset-til-flex). Denne typeovergangen oppretter en ny instans av et flex-objekt basert på Person-typen som delegerer til datastrømmen. Det resulterende flex-objektet blir sendt til flex-til-Infoset typeovergangen som initialiserer en instans av Person med data fra flex-objektet (flex-objektet befinner seg i strømmen siden flex-objektet delegerer). Som med serialisering, terminerer deserialiserings-pipelinen i en skriverfabrikk. Base-skriverfabrikken for deserialiserings-pipelinen har ansvar for å opprette instansen av den CLR-typen som blir deserialisert.

I tillegg til serialisering og deserialisering kan det være ønskelig å transformere Person-typen. Som angitt over omfatter Person-typens form to egenskaper: ForNavn

og EtterNavn. Anta for eksempel at en applikasjon som anvender denne definisjonen av Person vekselvirker med en annen applikajson som anvender en ulik definisjon av Person (f.eks. en Person med én egenskap – FulltNavn). Selv om én mulighet ville være å gjøre at begge applikasjonene anvender samme definisjon av Person-typen, vil ikke dette alltid være mulig (for eksempel kan begge applikasjonene allerede være skrevet).

I henhold til det utførelseseksempelet som beskrives kan det bli generert en typeovergang som konverterer formen til en Person-instans i én applikasjon til den formen som forventes av den andre. For å utføre transformasjonen (se figur 2), er det nødvendig å konstruere en ny flex-til-flex typeovergang (f.eks. brukerutvidelsesrutinen 260) og plassere denne i serialiserings-pipelinen etter den grunnleggende CLR-til-flex typeovergangen (f.eks standardrutinen 250). Under serialiseringsprosessen blir denne typeovergangen forsynt med et flex-objekt som delegerer til Person-instansen. Typeovergangen konstruerer en ny flex-type med den nye formen (én enkelt egenskap: FulltNavn). Basert på denne flex-typen blir det opprettet et nytt flex-objekt som sammenkjeder egenskapene ForNavn og Etternavn i det opprinnelige flex-objektet (som også delegerer til Person-instansen). Dette flex-objektet blir sendt til den grunnleggende flex-til-Infoset typeovergangen (f.eks. standardrutinen 270), som serialiserer én egenskap i stedet for to. Det er verdt å merke seg at den faktiske konkateneringen ikke trenger å bli gjennomført før flex-til-Infoset typeovergangen ber om verdien for den nye egenskapen FulltNavn. Følgelig utsettes konverteringen til opprettelse av Infoset-objektet eller det endelige objektet.

Følgelig kan en serialiseringsmotor i henhold til foreliggende oppfinnelse tilveieringe en utvidbar arkitektur for å konvertere mellom systemer og typer, som omfatter: støtte for pluggbare type- og datakonverteringer, støtte for foranderlige typer og objekter, støtte for pluggbare skjema-type systemer, støtte for pluggbare dataformater, etc.

Foreliggende oppfinnelse kan også beskrives med hensyn til fremgangsmåter som omfatter funksjonelle trinn og/eller ikke-funksjonelle handlinger. Det følgende er en beskrivelse av handlinger og trinn som kan utføres ved praktisering av foreliggende oppfinnelse. Vanligvis beskriver funksjonelle trinn oppfinnelsen med hensyn til

resultater som oppnås, mens ikke-funksjonelle handlinger beskriver mer spesifikke handlinger for å oppnå et ønsket resultat. Selv om de funksjonelle trinn og ikke-funksjonelle handlinger kan være beskrevet eller krevet i en gitt rekkefølge, er ikke foreliggende oppfinnelse nødvendigvis begrenset til noen konkret rekkefølge eller kombinasjon av handlinger og/eller trinn.

Figurene 5A-5B viser eksempler på handlinger og trinn i fremgangsmåter for å serialisere og deserialisere objekter i henhold til foreliggende oppfinnelse, som kan omfatte det å motta (512) et initielt objekt av en initiell type for kjøretidsbehandling av en serialiseringsmotor. Et trinn for å identifisere (520) typeinformasjon for det initielle objektet kan omfatte det å motta (522) typeinformasjonen. Typeinformasjonen kan forsynes i form av metadata assosiert med administrert kode. Et trinn for å konvertere (530) det initielle objektet til en mellomrepresentasjon basert på den informasjonen om den initielle typen kan omfatte det å generere (ikke vist) mellomrepresentasjonen basert på typeinformasjonen, anrope (532) én eller flere brukerdefinerte utvidelsesrutiner og anrope (534) én eller flere standardrutiner for å modifisere mellomrepresentasjonen. Den ene eller de flere utvidelsesrutinene endrer kjøremåten til serialiseringsmotoren.

Det skal bemerkes at mellomrepresentasjonen kan omfatte et objektnavn, en objekttype og/eller objektdata. Selv om det ikke er vist, kan et trinn for å modifisere (540) mellomrepresentasjonen også omfatte det å anrope (ikke vist) én eller flere brukerdefinerte utvidelsesrutiner og anrope (ikke vist) én eller flere standardrutiner for å modifisere mellomrepresentasjonen. Et trinn for å modifisere (540) mellomrepresentasjon kan videre omfatte det å endre (540) et objekts navn, type og/eller data. Et trinn for å utsette (550) modifiseringen kan omfatte det å spesifisere (552) hvordan mellomrepresentasjonen skal modifiseres uten faktisk å endre mellomrepresentasjonen. En slik utsettelse kan bidra til å redusere buffer- og prosesseringskravene som ellers er assosiert med det å modifisere mellomrepresentasjonen med en gang.

Et trinn for å konvertere (560) mellomrepresentasjonen av det initielle objektet til et endelig objekt av endelig type eller format kan omfatte følgende handlinger. Ved serialisering (563) kan trinnet omfatte det å opprette eller generere (565) det endelige

objektet. I ett utførelseseksempel som beskrevet ovenfor, formateres det endelige objektet i XML for transport. Det å opprette eller generere (565) det endelige objektet kan derfor omfatte det å generere den relevante XML-koden og skrive det endelige objektet til en datastrøm. Alternativt kan det endelige objektet bli formatert for persistent lagring til disk eller i et hvilket som helst annet format som er egnet for å representere det serialiserte initielle objektet. Ved deserialisering (564) kan trinnet omfatte det å instansiere (566) og initialisere (568) det endelige objektet. Under konverteringstrinnet (560) anropes brukerdefinerte utvidelsesrutiner og standard-rutiner for eventuelle utsatte modifikasjoner som har angitt hvordan en endring skal gjøres, men ikke faktisk utført endringen.

Utførelsesformer innenfor rammen av foreliggende oppfinnelse omfatter også datamaskinlesbare medier for å bære eller inneholde datamaskin-eksekverbare instruksjoner eller datastrukturer lagret deri. Slike datamaskinlesbare medier kan være et hvilket som helst tilgjengelig medium som kan aksesseres av en generell eller spesialinnrettet datamaskin. Som eksempler, uten begrensning, kan slike datamaskinlesbare medier omfatte RAM, ROM, EEPROM, CD-ROM eller andre optiske lagre, magnetdisk-lagre eller andre magnetiske lagringsanordninger, eller et hvilket som helst annet medium som kan anvendes for å bære eller lagre ønskede programkodeinnretninger i form av datamaskin-eksekverbare instruksjoner eller datastrukturer og som kan aksesseres av en generell eller spesialinnrettet data-maskin. Når informasjon blir overført eller levert over et nettverk eller en annen kommunikasjonsforbindelse (enten kablet, trådløs eller en kombinasjon av kablet og trådløs) til en datamaskin, betrakter datamaskinen på relevant måte forbindelsen som et datamaskinlesbart medium. En hvilken som helst slik forbindelse kan således betegnes et datamaskinlesbart medium. Kombinasjoner av det ovennevnte er også omfattet innenfor rammen av datamaskinlesbare medier. Datamaskin-eksekverbare instruksjoner kan for eksempel omfatte instruksjoner og data som forårsaker at en generell datamaskin, en spesialinnrettet datamaskin eller en spesialinnrettet prosesseringsanordning utfører en gitt funksjon eller gruppe av funksjoner.

Figur 6 og den følgende beskrivelsen er ment for å gi en kort, generell beskrivelse av et egnet databehandlingsmiljø i hvilket oppfinnelsen kan bli implementert.

Selv om det ikke er nødvendig, vil oppfinnelsen bli beskrevet i den generelle sammenhengen datamaskin-eksekverbare instruksjoner, så som programmoduler, som blir eksekvert av datamaskiner i nettverksmiljøer. Generelt omfatter programmoduler rutiner, programmer, objekter, komponenter, datastrukturer, etc. som utfører spesifikke oppgaver eller implementerer spesifikke abstrakte datatyper. Datamaskin-eksekverbare instruksjoner, assosierte datastrukturer og programmoduler representerer eksempler på programkodeinnretningene for å utføre trinn i fremgangsmåtene beskrevet her. Den konkrete sekvensen av slike eksekverbare instruksjoner eller assosierte datastrukturer representerer eksempler på motsvarende handlinger for å utføre funksjonene beskrevet i disse trinnene.

Fagmannen vil forstå at oppfinnelsen kan praktiseres i nettverkede databehandlingsmiljøer med mange typer datasystemstrukturer, omfattende personlige datamaskiner, håndholdte anordninger, flerprosessorsystemer, mikroprosessorbasert eller programmerbar forbrukerelektronikk, personlige datamaskiner i nettverk, minidatamaskiner, stormaskiner og liknende. Oppfinnelsen kan også praktiseres i distribuerte databehandlingsmiljøer der oppgaver blir utført av lokale og eksterne prosesseringsanordninger som er forbundet (enten via kablede forbindelser, trådløse forbindelser eller en kombinasjon av kablede og trådløse forbindelser) gjennom et kommunikasjonsnettverk. I et distribuert databehandlingsmiljø kan programmoduler befinne seg i både lokale og eksterne minnelagringsanordninger.

Som kan sees i figur 6, omfatter et eksempel på system for å implementere oppfinnelsen en generell databehandlingsanordning i form av en konvensjonell datamaskin 620 som omfatter en prosesseringsenhet 621, et systemminne 622 og en systembuss 623 som kopler forskjellige systemkomponenter inklusive systemminnet 622 til prosesseringsenheten 621. Systembussen 623 kan være en hvilken som helst av mange typer busstrukturer, omfattende en minnebuss eller minnekontroller, en ekstern buss og en lokal buss, som anvender en hvilken som helst av en rekke tilgjengelige bussarkitekturer. Systemminnet omfatter et leseminne (ROM) 624 og et direkteaksessminne (RAM) 625. Et BIOS (basic input/output system) 626 som inneholder de grunnleggende rutinene som hjelper til å overføre informasjon mellom

elementer i datamaskinen 620, for eksempel under oppstart, kan være lagret i ROM 624.

Datamaskinen 620 kan også omfatte en harddiskstasjon 627 for å lese fra og skrive til en magnetisk harddisk 639, en magnetdisk-stasjon 628 for å lese fra eller skrive til en flyttbar magnetdisk 629 og en optisk stasjon 630 for å lese fra eller skrive til en flyttbar optisk disk 631, så som et CD-ROM eller et annet optisk medium. Harddiskstasjonen 627, magnetdisk-stasjonen 628 og den optiske stasjonen 630 er koplet til systembussen 623 henholdsvis via et harddiskstasjon-grensesnitt 632, et magnetdiskstasjon-grensesnitt 633 og et optisk stasjon-grensesnitt 634. Stasjonene og deres assosierte datamaskinlesbare medier tilveiebringer ikke-volatil lagring av datamaskin-eksekverbare instruksjoner, datastrukturer, programmoduler og andre data for datamaskinen 620. Selv om det eksemplifiserte miljøet som beskrives her omfatter en magnetisk harddisk 639, en flyttbar magnetisk disk 629 og en flyttbar optisk disk 631, kan andre typer datamaskinlesbare medier for å lagre data anvendes, inklusive magnetkassetter, flashminnekort, DVD-plater, Bernoulli-patroner, RAM-minner, ROM-minner og liknende.

Programkodeinnretninger som omfatter én eller flere programmoduler kan være lagret i harddisken 639, magnetdisken 629, den optiske disken 631, ROM 624 eller RAM 625, inklusive et operativsystem 635, ett eller flere applikasjonsprogrammer 636, andre programmoduler 637 og programdata 638. En bruker kan mate inn kommandoer og informasjon til datamaskinen 620 via et tastatur 640, en pekeranordning 642 eller andre innmatingsanordninger (ikke vist), så som en mikrofon, en styrespak, en spillkontroll, en parabolantenne, en skanner eller liknende. Disse og andre innmatingsanordninger er ofte koplet til prosesseringsenheten 621 via et serieportgrensesnitt 646 som er koplet til systembussen 623. Alternativt kan innmatingsanordningene være tilkoplet via andre grensesnitt, så som en parallellport, en spillutgang eller en universell seriell buss (USB-port). En monitor 647 eller en annen type fremvisningsanordning er også koplet til systembussen 623 via et grensesnitt, så som et skjermkort 648. I tillegg til monitoren omfatter personlige datamaskiner typisk andre periferiske utmatingsanordninger (ikke vist), så som høyttalere og skrivere.

Datamaskinen 620 kan kjøre i et nettverket miljø som anvender logiske forbindelser til én eller flere fjerndatamaskiner, så som fjerndatamaskiner 649a og 649b. Fjerndatamaskinene 649a og 649b kan hver være en annen personlig datamaskin, en tjener, en ruter, en nettverks-PC, en peer-anordning eller en annen vanlig nettverksnode, og omfatter typisk mange av eller alle de elementene som er beskrevet ovenfor i forbindelse med datamaskinen 620, selv om kun minnelagringsanordninger 650a og 650b og deres assosierte applikasjonsprogrammer 636a og 636b er illustrert i figur 6. De logiske forbindelsene som er vist i figur 6 omfatter et lokalt nettverk (LAN) 651 og et regionalt nettverk (WAN) 652, som er vist her kun som eksempler og ikke begrensning. Slike nettverksmiljøer er vanlige i kontornettverk, bedriftsomspennende datanettverk, intranett og Internett.

Når den blir anvendt i et LAN-nettverksmiljø, er datamaskinen 620 koplet til det lokale nettverket 651 via et nettverksgrensesnitt eller nettverkskort 653. Når den blir anvendt i et WAN-nettverksmiljø, kan datamaskinen 620 omfatte et modem 654, en trådløs forbindelse eller andre midler for å etablere kommunikasjon over det regionale nettverket 652, for eksempel Internett. Modemet 654, som kan være internt eller eksternt, er koplet til systembussen 623 via serieport-grensesnittet 646. I et nettverksmiljø kan programmoduler som er vist i forbindelse med datamaskinen 620, eller deler av disse, være lagret i den eksterne minnelagringsanordningen. En vil forstå at de viste nettverksforbindelsene kun er eksempler, og at andre midler for å etablere kommunikasjon over det regionale nettverket 652 kan anvendes.

Foreliggende oppfinnelse kan realiseres i andre konkrete former uten at en fjerner seg fra dens idé eller essensielle særtrekk. De beskrevne utførelsesformene skal i alle henseende betraktes som kun illustrerende, og ikke begrensende. Oppfinnelsens ramme defineres derfor av de etterfølgende kravene heller enn av den foregående beskrivelsen. Alle endringer som faller innenfor kravenes betydning og ekvivalensramme skal være omfattet av disse.

PATENTKRAV

1. Fremgangsmåte, som utføres i et datasystem som omfatter en utvidbar serialiseringsmotor som serialiserer og deserialiserer dataobjekter av forskjellig type, for å konvertere et initielt objekt av en initiell type til et endelig objekt av en endelig type, der fremgangsmåten muliggjør endring av kjøremåten til serialiseringsmotoren gjennom én eller flere utvidelsesrutiner uten at det er nødvendig å bytte ut én eller flere eksisterende rutiner i serialiseringsmotoren, idet fremgangsmåten omfatter trinn for å:

identifisere typeinformasjon for et initielt objekt av en initiell type som mottas for kjøretidsprosessering av serialiseringsmotoren,

basert på typeinformasjonen, konvertere det initielle objektet til en mellomrepresentasjon av det initielle objektet som er egnet for kjøretidsmodifisering,

modifisere mellomrepresentasjonen av det initielle objektet i henhold til én eller flere utvidelsesrutiner, og med det endre kjøremåten til serialiseringsmotoren,

konvertere mellomrepresentasjonen av det initielle objektet til et endelig objekt av en endelig type, og

å utsette modifisering av mellomrepresentasjonen til mellomrepresentasjonen konverteres til det endelige objektet for å unngå å måtte bufre en modifisering av mellomrepresentasjonen.

2. Fremgangsmåte ifølge krav 1, der mellomrepresentasjonen av det initielle objektet omfatter minst ett av et objektnavn, en objekttype og objektdata.

3. Fremgangsmåte ifølge krav 2, der trinnet for å endre mellomrepresentasjonen av det initielle objektet i henhold til én eller flere utvidelsesrutiner omfatter det å modifisere minst ett av objektnavnet, objekttypen og objektdataene.

4. Fremgangsmåte ifølge krav 1, det å modifisere mellomrepresentasjonen av det initielle objektet baseres på enten en gitt modell eller struktur i typeinformasjonen, objektdata i det initielle objektet eller begge deler.

5. Fremgangsmåte ifølge krav 1, der det initielle objektet omfatter et minnelagret objekt, og serialiseringsmotoren serialiserer det initielle objektet for å generere det endelige objektet.

6. Fremgangsmåte ifølge krav 1, der det endelige objektet omfatter et minnelagret objekt som skal instansieres og initialiseres basert på det initielle objektet, og der serialiseringsmotoren deserialiserer det initielle objektet for å skape det endelige objektet.

7. Fremgangsmåte ifølge krav 1, der både det initielle objektet og det endelige objektet er minnelagrede objekter.

8. Dataprogramprodukt som implementerer en utvidbar serialiseringsmotor for å konvertere ett eller flere initielle objekter av én eller flere initielle typer til ett eller flere endelige objekter av én eller flere endelige typer, der kjøremåten til serialiseringsmotoren kan bli endret uten at det er nødvendig å re-implementere eksisterende deler av serialiseringsmotoren, idet dataprogramproduktet omfatter ett eller flere datamaskinlesbare medier som bærer datamaskin-eksekverbare instruksjoner i form av programmoduler, der programmodulene omfatter:

en refleksjonsmodul som kan bli byttet ut under kjøring, for å identifisere typeinformasjon for et initielt objekt av en initiell type som er mottatt for prosessering av serialiseringsmotoren,

én eller flere konverteringsmoduler som kan bli byttet ut under kjøring, for å generere og modifisere en mellomrepresentasjon av det initielle objektet basert på den identifiserte typeinformasjonen, idet den ene eller de flere kjøretid-utskiftbare konverteringsmodulene omfatter én eller flere utvidelsesrutiner som endrer kjøremåten til serialiseringsmotoren, og

en genereringsmodul som kan bli byttet ut ved kjøretid, for å opprette et endelig objekt av en endelig type basert på mellomrepresentasjonen generert av konverteringsmodulen, hvor den ene eller de flere kjøretid-utskiftbare konverteringsmodulene er i stand til å utsette én eller flere endringer av mellomrepresentasjonen av det initielle objektet inntil mellomrepresentasjonen blir konvertert til det endelige objektet for å unngå bufferkrav i forbindelse med det å utføre den ene eller de flere modifiseringene av mellomrepresentasjonen.

9. Dataprogramprodukt ifølge krav 8, der mellomrepresentasjonen omfatter et objektnavn, en objekttype og objektdata for det initielle objektet og eventuelle objekter inneholdt i det initielle objektet.

10. Dataprogramprodukt ifølge krav 8, der den ene eller de flere kjøretid-utskiftbare konverteringsmodulene er i stand til å endre minst én av et objektnavn, en objekttype og objektdata for det initielle objektet og eventuelle objekter inneholdt i det initielle objektet.

11. Dataprogramprodukt ifølge krav 8, der den ene eller de flere kjøretid-utskiftbare konverteringsmodulene er i stand til å holde rede på informasjon vedrørende det initielle objektet uten å modifisere mellomrepresentasjonen.

12. Dataprogramprodukt ifølge krav 8, der den ene eller de flere kjøretid-utskiftbare konverteringsmodulene er i stand til å modifisere mellomrepresentasjonen av det initielle objektet basert på enten en gitt modell eller struktur i typeinformasjonen, objektdata i det initielle objektet, eller begge deler.

13. Dataprogramprodukt ifølge krav 8, der den kjøretid-utskiftbare genereringsmodulen er i stand til å opprette det endelige objektet i XML-format.

14. Dataprogramprodukt ifølge krav 8, der den kjøretid-utskiftbare genereringsmodulen er i stand til å instansiere og initialisere det endelige objektet basert på mellomrepresentasjonen.

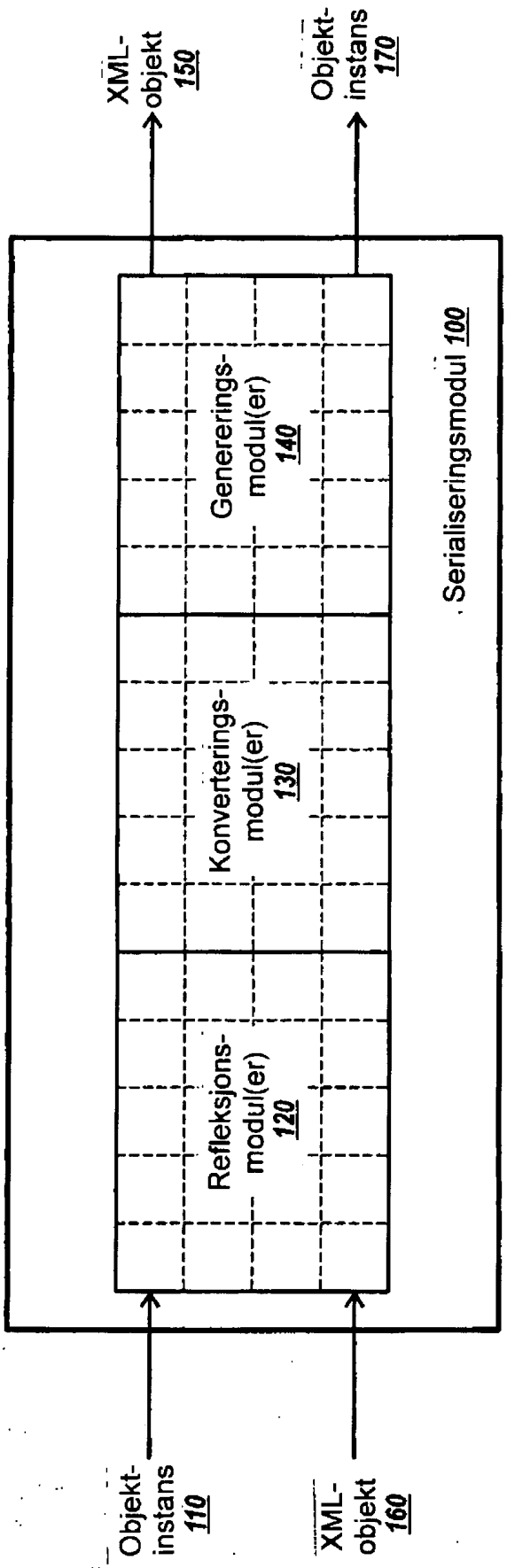

XML-objekt 150
Objekt-instans 170
Genereringsmodul(er) 140
Konverteringsmodul(er) 130
Refleksjonsmodul(er) 120
Serialiseringsmodul 100
Objekt-instans 110
XML-objekt 160

Fig. 1

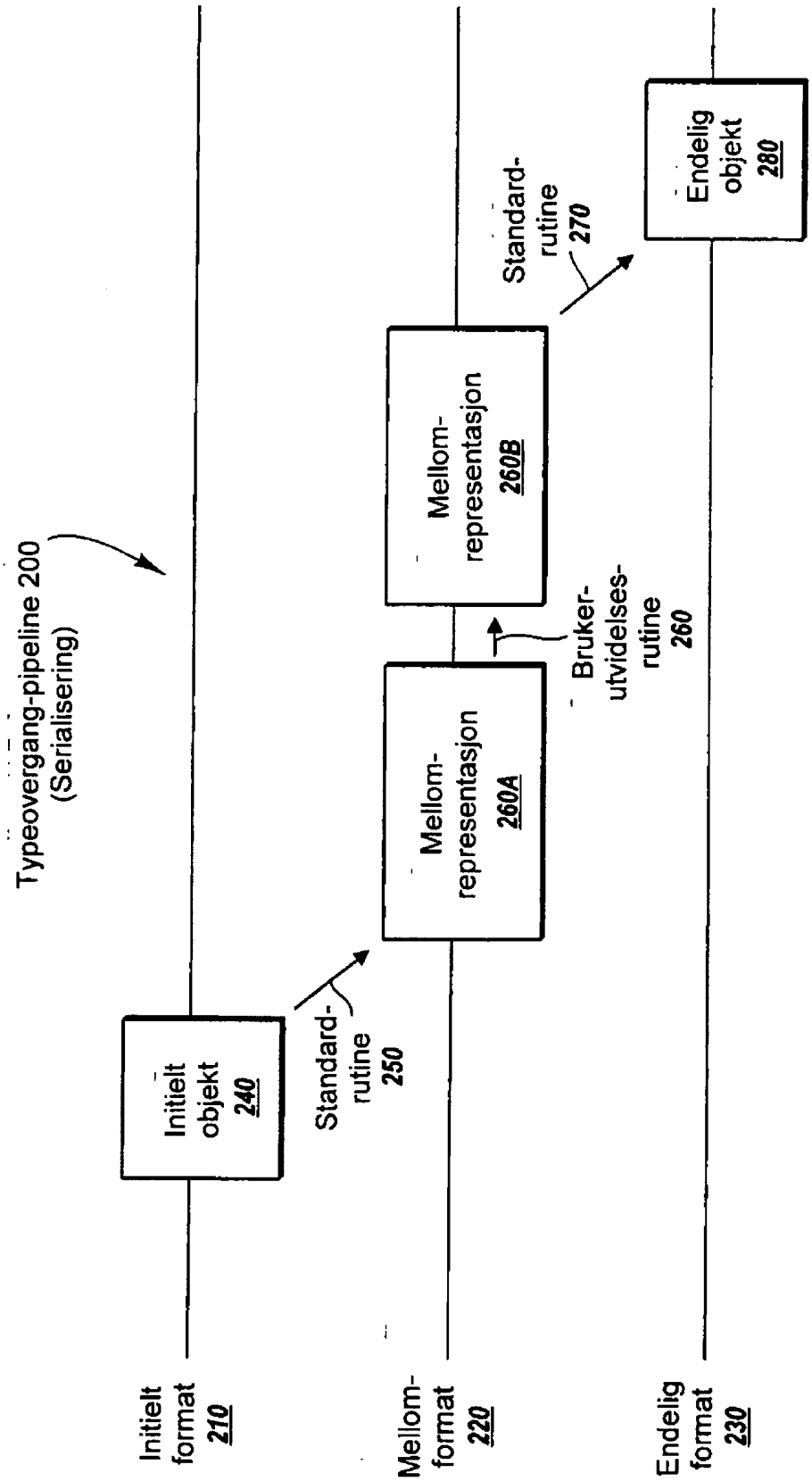

Typeovergang-pipeline 200
(Serialisering)
Initielt objekt 240
Standard-rutine 250
Mellom-representasjon 260A
Bruker-utvidelses-rutine 260
Mellom-representasjon 260B
Standard-rutine 270
Endelig objekt 280
Initielt format 210
Mellom-format 220
Endelig format 230

Fig. 2

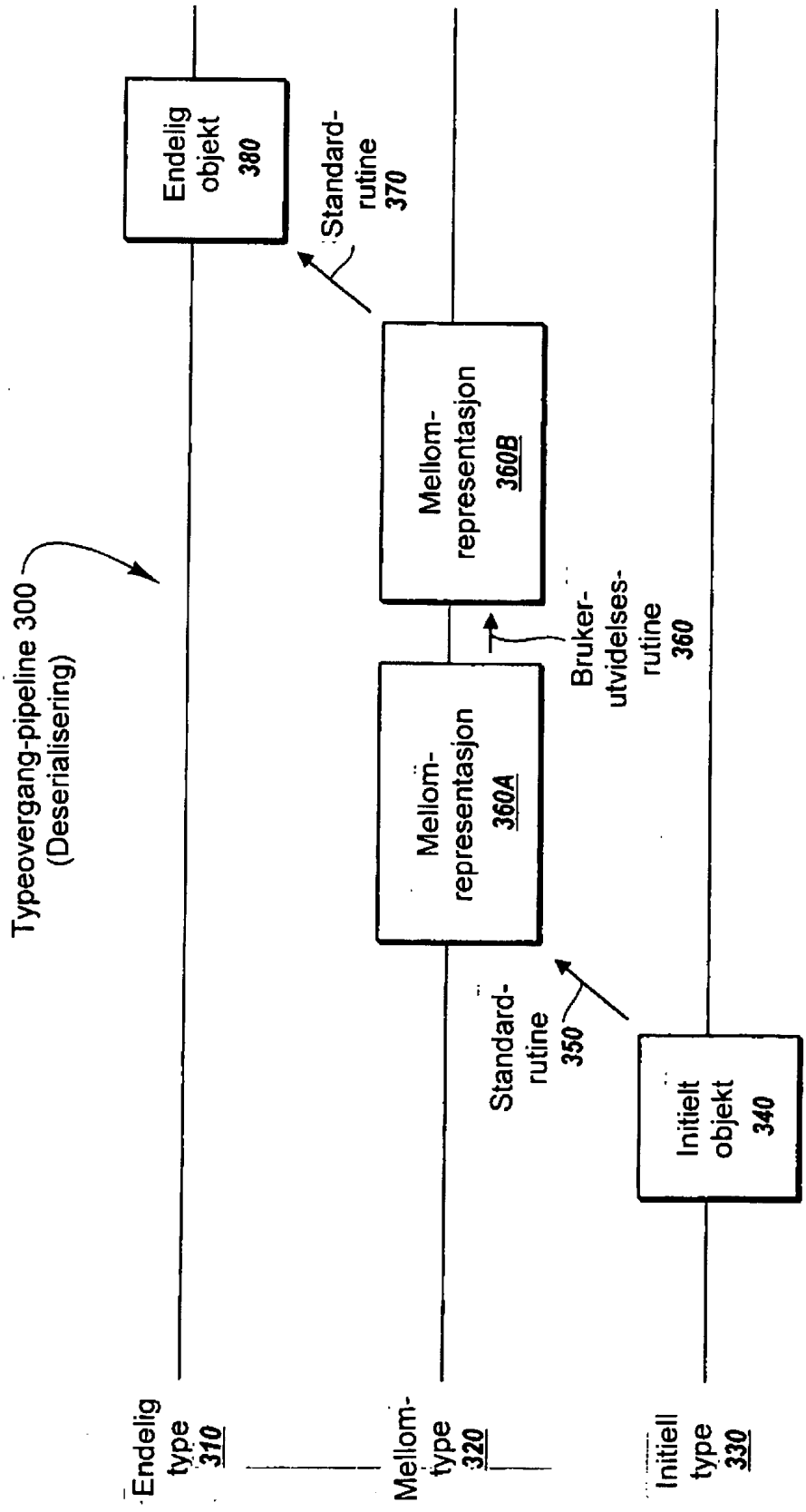

Endelig type 310
Mellom-type 320
Initiell type 330
Typeovergang-pipeline 300 (Deserialisering)
Endelig objekt 380
Standard-rutine 370
Mellom-representasjon 360B
Bruker-utvidelses-rutine 360
Mellom-representasjon 360A
Standard-rutine 350
Initielt objekt 340

Fig. 3

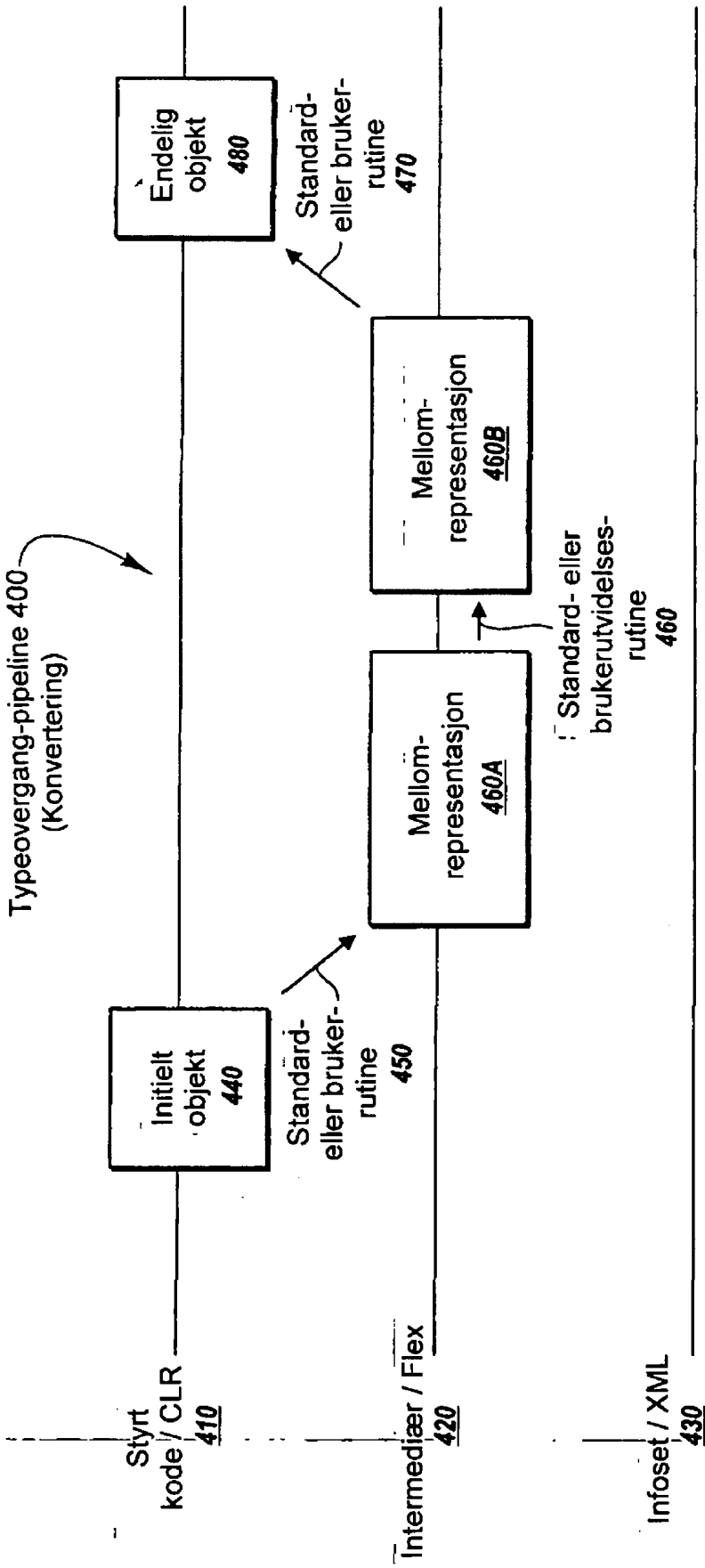

Typeovergang-pipeline 400
(Konvertering)
Styrt kode / CLR 410
Intermediær / Flex 420
Infoset / XML 430
Initielt objekt 440
Standard- eller bruker-rutine 450
Mellom-representasjon 460A
Standard- eller brukerutvidelses-rutine 460
Mellom-representasjon 460B
Standard- eller bruker-rutine 470
Endelig objekt 480

Fig. 4

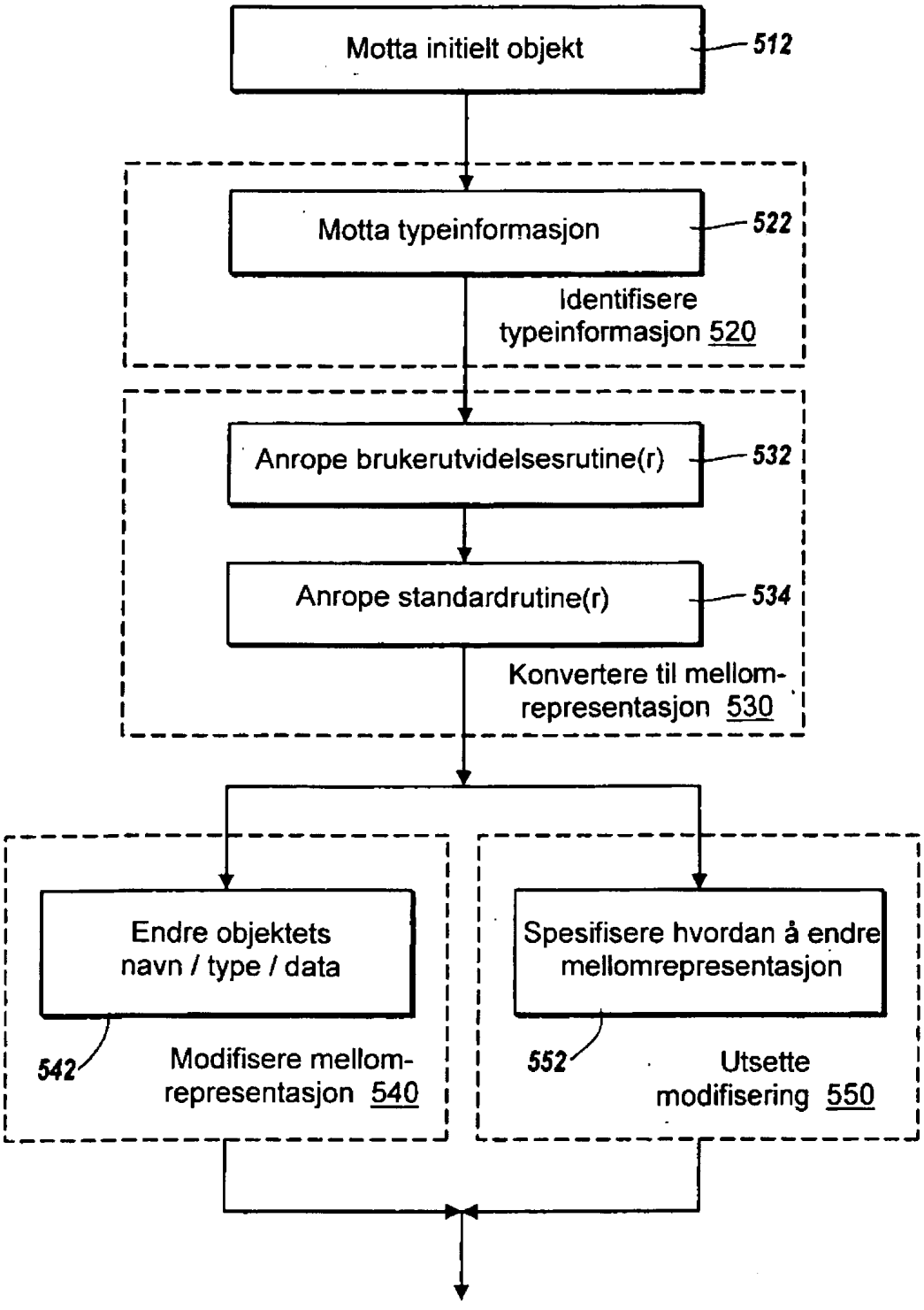

Motta initielt objekt
512
Motta typeinformasjon
522
Identifisere typeinformasjon 520
Anrope brukerutvidelsesrutine(r)
532
Anrope standardrutine(r)
534
Konvertere til mellom-representasjon 530
Endre objektets navn / type / data
542
Modifisere mellom-representasjon 540
Spesifisere hvordan å endre mellomrepresentasjon
552
Utsette modifisering 550
Til figur 5B

Fig. 5A

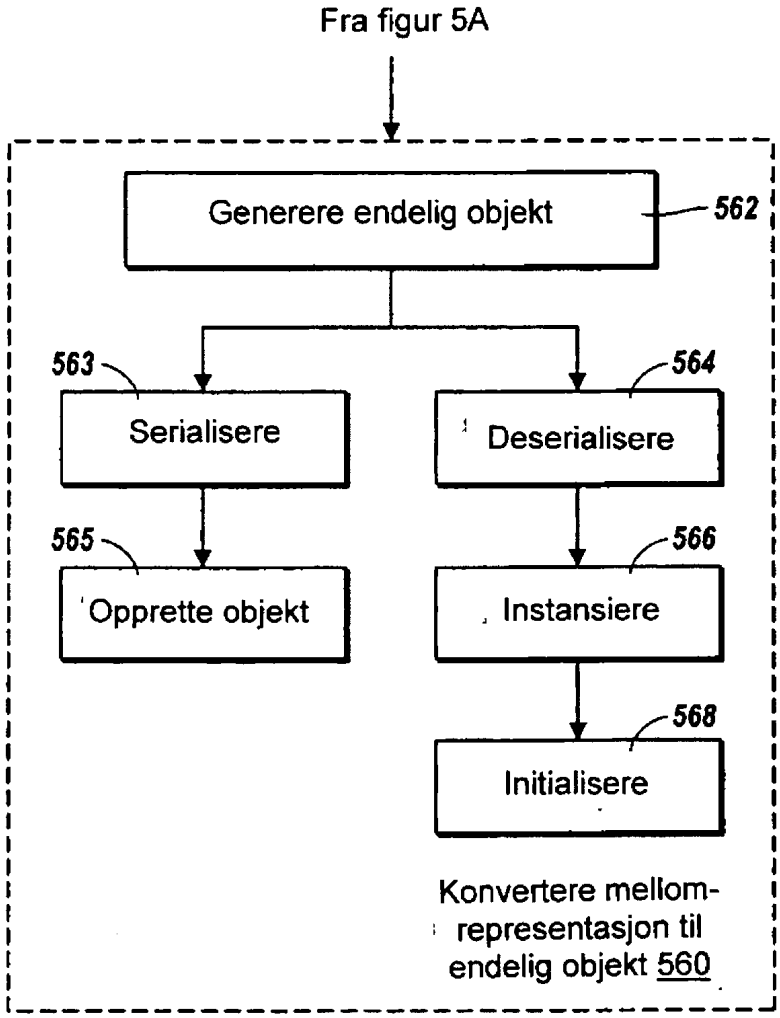

Fra figur 5A
Generere endelig objekt
562
563
Serialisere
564
Deserialisere
565
Opprette objekt
566
Instansiere
568
Initialisere
Konvertere mellom-representasjon til endelig objekt 560

Fig. 5B

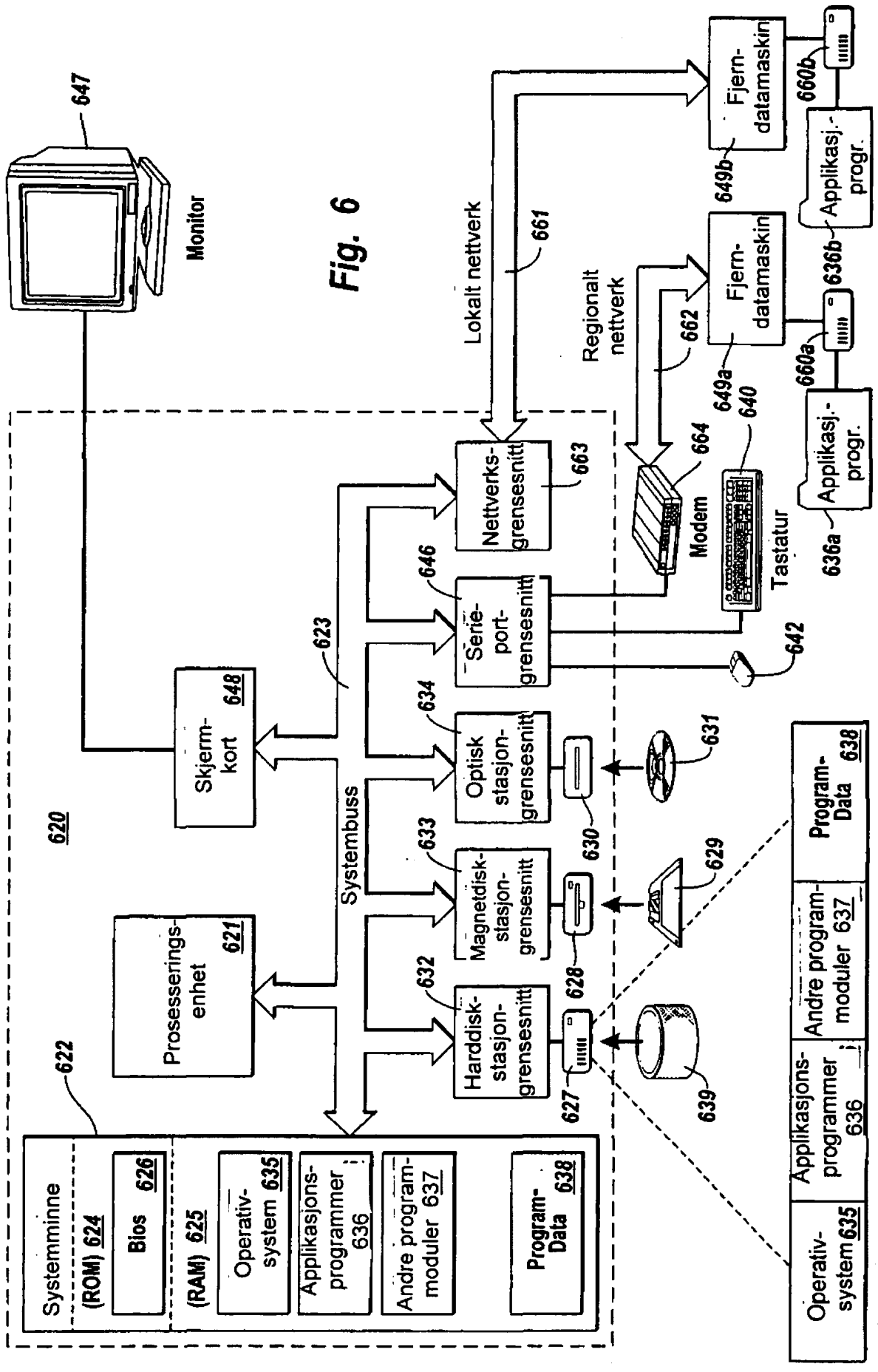

647
Monitor
Lokalt nettverk
661
Regionalt nettverk
662
Fjern-datamaskin
649b
660b
Applikasj.-progr.
Fjern-datamaskin
649a
636b
660a
640
664
Modem
Tastatur
636a
Applikasj.-progr.
Nettverks-grensesnitt
663
646
Serie-port-grensesnitt
642
623
Skjerm-kort
648
634
Optisk stasjon-grensesnitt
631
620
Systembuss
633
Magnetdisk-stasjon-grensesnitt
630
629
Prosesserings-enhet
621
632
Harddisk-stasjon-grensesnitt
628
639
627
622
Systemminne
(ROM) 624
Bios 626
(RAM) 625
Operativ-system 635
Applikasjons-programmer 636
Andre program-moduler 637
Program-Data 638
Program-Data 638
Andre program-moduler 637
Applikasjons-programmer 636
Operativ-system 635

Fig. 6