(12) **PATENT**

(19) NO (11) **329240** (13) **B1**

NORGE

(51) Int Cl.
G06F 17/22 (2006.01)
G06F 9/45 (2006.01)
G06F 9/44 (2006.01)

Patentstyret

(21)	Søknadsnr	20032232	(86)	Int.inng.dag og søknadsnr	
(22)	Inng.dag	2003.05.16	(85)	Videreføringsdag	
(24)	Løpedag	2003.05.16	(30)	Prioritet	2003.02.28, US, 377313
(41)	Alm.tilgj	2004.08.30			
(45)	Meddelt	2010.09.20			

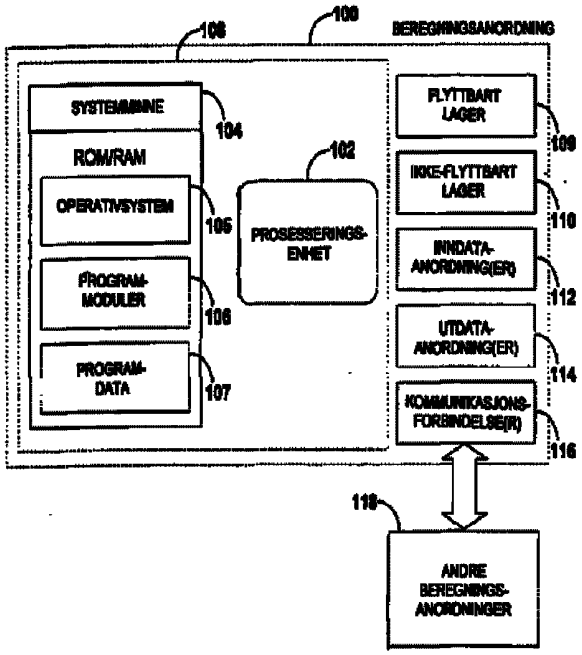
(73) Innehaver Microsoft Corp, One Microsoft Way, US-WA98052-6399 REDMOND, USA
(72) Oppfinner Sundaram Ramani, 14116 Northeast 85th Court, Redmond, WA 98052, US-, USA
Robert A Relyea, 4618 142nd Place Southeast, Bellevue, WA 98006, US-, USA
Jeffrey L Bogdan, 6308 East Lake Sammamish Parkway Northeast, #212, Redmond, WA 98052, US-, USA
(74) Fullmektig Bryn Aarflot AS, Postboks 449 Sentrum, 0104 OSLO, Norge

(54) Benevnelse **System og fremgangsmåte for forklarende definering og bruk av undergrupper innenfor dokumentkoding**

(56) Anførte publikasjoner US 2003/037311

(57) Sammendrag

Det beskrives et system, en fremgangsmåte samt en datastruktur som gjør det mulig å generere en eksekverbar assemblering assosiert med en underklasse fra en underklasse-definisjon inne i et formatkodet dokument. Ifølge oppfinnelsen blir underklasse-definisjonen skrevet basert på et skjema. Skjemaet kan være XML-basert. Skjemaet omfatter en underklasse-etikett for å definere et navn for underklassen. Navnet blir assosiert med en type for et objekt som blir instansiert når den eksekverbare assembleringen eksekveres. Skjemaet omfatter videre ett eller flere hint, for eksempel for å spesifisere et programmeringsspråk for å kompilere underklasse-definisjonen, for å spesifisere en baseklasse fra hvilken underklassen avledes, for å spesifisere handlinger å utføre når objektet blir instansiert, for å opprette en hendelse-definisjon og en hendelsehåndterer for underklassen og for å spesifisere en egenskap som blir en medlemsvariabel i objektet når objektet instansieres.



I dag gjør programvareutviklingsverktøy det mulig for programvareutviklere å bygge eksekverbare komponenter ved anvendelse av ett eller flere programmeringsspråk, så som C, C++, C# og liknende. Én fordel ved å bygge eksekverbare komponenter er at komponentene, når de er bygget, kan gjenbrukes av andre programmer. En annen fordel ved å bygge eksekverbare komponenter er at nye komponenter på en enkel måte kan avledes fra eksisterende komponenter.

Generelt blir komponenter utvidet ved å implementere en underklasse, hvilket betyr å avlede en ny klasse fra en eksisterende klasse. Disse klassene og underklassene blir skrevet ved anvendelse av ett av programmeringsspråkene. Koden som skrives blir vanligvis betegnet kildekode. I tradisjonelle kjøremiljøer kompilerer programvareutviklingsverktøyene kildekoden til objektkode og linker deretter flere objektkoder sammen for å skape en eksekverbar fil. Ett av problemene med disse tradisjonelle kjøremiljøene er imidlertid det at hvert programmeringsspråk og hver versjon av programmeringsspråket krever et forskjellig kjøremiljø.

For å overkomme dette problemet har det blitt utviklet en ny type kjøremiljø som effektivt eliminerer mange av problemene med grensesnittet mellom språkene og språkversjonene i de tradisjonelle kjøremiljøene. I denne nye typen kjøremiljø kompilerer utviklingsverktøy kildekoden til et mellomspråk. Ved kjøretid kompilerer kjøremiljøet mellomspråket til intern binær, eksekverbar kode. Det nye kjøremiljøet utfører således "linke"-prosessen ved kjøretid. For å utføre denne "linke-type" prosessen, leser kjøremiljøet informasjon (f.eks. metadata) og aksesserer IL-assembleringer for de komponentene som er assosiert med det programmet som kjøres. Metadataene omfatter beskrivelser av typer, versjoner, ressurser og liknende. IL-assembleringene kan være ett enkelt dynamisk linket bibliotek (DLL) eller flere DLL-biblioteker og ressurser.

I både det tradisjonelle og den nye typen kjøremiljø blir kildekode skrevet ved anvendelse av et programmeringsspråk. Hvert programmeringsspråk har sin egen unike syntaks og sitt eget sett av API (application programming interface)-funksjoner som er spesifikke for kjøremiljøet. For at en programmerer skal kunne skrive kildekode, må utvikleren først lære seg programmeringsspråkets syntaks og de API-funksjonene som er assosiert med kjøremiljøet. Det å lære

seg syntaksen og API-funksjonene er veldig tidkrevende og utfordrende. I tillegg, dersom en utvikler ønsker å programmere i flere programmeringsspråk og/eller forskjellige kjøremiljøer, må utvikleren huske likhetene og de subtile forskjellene mellom hvert av programmeringsspråkenes syntaks og API-funksjonene for de forskjellige kjøremiljøene.

US 2003/0037311 A1 beskriver en fremgangsmåte og apparat som anvender datamaskin script-språk i multimedia anvendelsesplattformer. Det datamaskinimplementerte systemet og fremgangsmåten utfører en mengde forskjellige oppgaver relatert til skapelsen, forvaltningen og presentasjonen av multimediainnhold.

Gitt fordelene ved å anvende komponenter, er det behov for en bedre mekanisme for å bygge, utvide og anvende komponenter.

I et første aspekt tilveiebringer oppfinnelsen et datamaskinlesbart medium innkodet med en datamaskinlesbar datastruktur, der datastrukturen omfatter:

et skjema for forklarende definering av en underklasse-definisjon i et formatkodet dokument, idet skjemaet kan bli kompilert til en eksekverbar assemblering som instansierer et objekt som er assosiert med en underklasse som er definert innenfor underklasse-definisjonen ved eksekvering av assembleringen, hvor den eksekverbare assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den eksekverbare sammenstillingen er kompilert uten hensyn til programmeringsspråket.

I et andre aspekt tilveiebringer oppfinnelsen et datamaskinlesbart medium som inneholder instruksjoner for å generere en eksekverbar assemblering for en underklasse, der instruksjonene omfatter det å:

identifisere en underklasse-definisjon i et formatkodet dokument, idet underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan bli eksekvert for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen

slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

I et tredje aspekt tilveiebringer oppfinnelsen en datamaskin-implementert fremgangsmåte for å generere en eksekverbar assemblering, idet fremgangsmåten omfatter det å:

identifisere en underklasse-definisjon i et formatkodet dokument, der underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan eksekveres for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

I et fjerde aspekt tilveiebringer oppfinnelsen et datamaskinlesbart medium som omfatter instruksjoner for å skape en eksekverbar assemblering for en underklasse, idet instruksjonene omfatter fremgangsmåten angitt over.

I et femte aspekt tilveiebringer oppfinnelsen et datasystem for å generere en assemblering fra en underklasse-definisjon med et formatkodet dokument, der datasystemet omfatter:

en prosessor; og

et minne, der minnet er allokert for flere datamaskin-eksekverbare instruksjoner som blir lastet inn i minnet, hvor prosessoren eksekverer instruksjonene for å:

identifisere en underklasse-definisjon i et formatkodet dokument, idet underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan bli eksekvert for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

I et sjette aspekt tilveiebringer oppfinnelsen en datamaskin-implementert fremgangsmåte for å generere en eksekverbar assemblering, idet fremgangsmåten omfatter:

et middel for å identifisere en underklasse-definisjon i et formatkodet dokument, der underklasse-definisjonen definerer en underklasse; og

et middel for å generere en assemblering basert på underklasse-definisjonen, der assembleringen kan eksekveres for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

Foreliggende oppfinnelse er rettet mot et system og en fremgangsmåte for forklarende definering, utvidelse og bruk av underklasser innenfor dokumentkode. Oppfinnelsen tilveiebringer en mekanisme for utviklere til å bygge, utvide og anvende komponenter ved anvendelse av et formatteringsspråk. Disse komponentene omfatter komponenter som kan gjenbrukes, API-funksjoner, dokument-brukergrensesnitt og liknende. Mekanismen krever ikke at en utvikler kan et programmeringsspråk. I stedet gjør mekanismen det mulig for brukeren å anvende et kjent formatteringsspråk, så som XML (extensible markup language), for å bygge komponenter. Ettersom XML er mye enklere å lære, og er i ferd med å bli mer kjent innenfor de generelle dataprogrammeringsmiljøene, tilbyr foreliggende oppfinnelse mange fordeler fremfor det å anvende tradisjonelle programmeringsspråk. Én fordel er at komponenter kan bli definert i et formatkodet dokument, sammen med annen formatkodet tekst, for å skape et veldig sofistikert elektronisk dokument. En annen fordel er at utviklere ikke trenger å kunne eller forstå noe programmeringsspråk for å generere en eksekverbar komponent.

Systemet, fremgangsmåten og datastrukturen ifølge foreliggende oppfinnelse gjør det mulig å generere en eksekverbar assemblering assosiert med en underklasse fra en underklasse-definisjon som er skrevet i et formatkodet dokument. Ifølge oppfinnelsen skrives underklasse-definisjonen basert på et skjema. Skjemaet kan være XML-basert. Skjemaet omfatter en underklasse-etikett for å definere et navn for underklassen. Navnet blir assosiert med en type for et

objekt som blir instansiert når den eksekverbare assembleringen blir eksekvert. Skjemaet omfatter videre ett eller flere hint, for eksempel for å spesifisere et programmeringsspråk for å kompilere underklasse-definisjonen, for å spesifisere en baseklasse fra hvilken underklassen skal avledes, for å spesifisere handlinger å utføre når objektet blir instansiert, for å opprette en hendelse-definisjon og en assosiert hendelsehåndterer for underklassen og for å spesifisere en egenskap som blir en medlemsvariabel i objektet når objektet instansieres.

Figur 1 illustrerer et eksempel på beregningsanordning som kan anvendes i forklarende implementasjoner av foreliggende oppfinnelse.

Figur 2 er et funksjonelt blokkdiagram som illustrerer generelt komponenter for å implementere én utførelsesform av foreliggende oppfinnelse.

Figur 3 er et funksjonelt blokkdiagram som illustrerer generelt et kjøremiljø for å implementere én utførelsesform av foreliggende oppfinnelse.

Figurene 4-6 illustrerer en serie av hoveddeler av formatkodede dokumenter som illustrerer et eksempel på syntaks for forklarende definering av underklasser ifølge én utførelsesform av foreliggende oppfinnelse.

Figur 7 illustrerer hoveddeler av et formatkodet dokument som illustrerer et eksempel på syntaks for å anvende en eksekverbar komponent fra inne i et formatkodet dokument.

Figur 8 er et logisk flytdiagram som illustrerer generelt en prosess for å kompilere deklarativt definerte underklasser ifølge én utførelsesform av foreliggende oppfinnelse.

Figur 9 er et logisk flytdiagram som illustrerer generelt en kjøretidprosess for anvendelse av underklasser som er deklarert i et formatkodet dokument ifølge én utførelsesform av foreliggende oppfinnelse.

Figur 10 er et kildekodeeksempel som lister motsvarende kildekode som er generert av formatteringsspråk-kompilatoren vist i figur 2 basert på formatkodede dokumenter illustrert i figurene 4-6.

Foreliggende oppfinnelse er rettet mot et system og en fremgangsmåte for deklarativ definering, utvidelse og bruk av underklasser i et formatkodet dokument. Oppfinnelsen tilveiebringer en mekanisme for utviklere til å bygge, utvide og bruke komponenter ved anvendelse av et formatteringsspråk. Mekanismen krever ikke at en utvikler kan et programmeringsspråk. I stedet gjør meka-

nismen det mulig for brukeren å anvende et kjent formatteringsspråk, så som XML (extensible markup language), for å bygge komponenter.

Illustrativt beregningsmiljø

Figur 1 illustrerer et eksempel på beregningsanordning som kan anvendes i forklarende implementasjoner av foreliggende oppfinnelse. Med henvisning til figur 1, i en helt grunnleggende konstruksjon, omfatter beregningsanordningen 100 typisk minst én prosesseringsenhet 102 og et systemminne 104. Avhengig av den eksakte konstruksjonen og typen av beregningsanordning 100, kan systemminnet være volatilt (så som RAM), ikke-volatilt (så som ROM, flash-minne, etc.) eller en kombinasjon av de to. Systemminnet 104 omfatter typisk et operativsystem 105, én eller flere programmoduler 106, og kan omfatte programdata 107. Eksempler på programmoduler 106 omfatter en browser-applikasjon, en finansadministreringsapplikasjon, en tekstbehandler og liknende. Denne grunnleggende konstruksjonen er illustrert i figur 1 av komponentene innenfor den stiplede boksen 108.

Beregningsanordningen 100 kan omfatte ytterligere mekanismer eller funksjonalitet. For eksempel kan beregningsanordningen 100 også omfatte ytterligere datalagringsanordninger (flyttbare og/eller ikke-flyttbare) så som, for eksempel, magnetdisker, optiske disker eller bånd. Slike ytterligere lagre er illustrert i figur 1 av et flyttbart lager 109 og et ikke-flyttbart lager 110. Datamaskinlagringsmedier kan omfatte volatile og ikke-volatile, flyttbare og ikke-flyttbare medier som er implementert med en hvilken som helst fremgangsmåte eller teknologi for lagring av informasjon, så som datamaskinlesbare instruksjoner, datastrukturer, programmoduler eller andre data. Systemminnet 104, det flyttbare lageret 109 og det ikke-flyttbare lageret 110 er alle eksempler på datamaskin-lagringsmedier. Datamaskin-lagringsmedier omfatter, men er ikke begrenset til, RAM, ROM, EEPROM, flash-minne eller annen minneteknologi, CD-ROM, DVD eller annen optisk lagring, magnetkasetter, magnetbånd, magnetdisklagre eller andre magnetiske lagringsanordninger, eller et hvilket som helst annet medium som kan anvendes for å lagre den ønskede informasjonen og som kan aksesseres av beregningsanordningen 100. Ethvert slikt datamaskin-lagringsmedium kan være en del av anordningen 100. Beregningsanordningen 100 kan

også ha én eller flere inndataanordninger 112 så som tastatur, mus, penn, stemmeinnmatingsanordning, berøring-innmatingsanordning, etc. Én eller flere utdataanordninger 114, så som en monitor, høyttalere, en skriver etc. kan også være inkludert. Disse anordningene er velkjente for fagmannen, og trenger ikke diskuteres utførlig her.

Beregningsanordningen 100 kan også omfatte kommunikasjonsforbindelser 116 som gjør det mulig for anordningen 100 å kommunisere med andre beregningsanordninger 118, for eksempel over et nettverk. Kommunikasjonsforbindelsene 116 er ett eksempel på kommunikasjonsmedium. Kommunikasjonsmedier kan typisk være innlemmet i datamaskinlesbare instruksjoner, datastrukturer, programmoduler eller andre data i et modulert datasignal, så som er bærerbølge eller en annen transportmekanisme, og omfatter ethvert informasjonsleveringsmedium. Betegnelsen "modulert datasignal" betyr et signal som får én eller flere av sine egenskaper satt eller endret på en slik måte at det innkodes informasjon i signalet. Som et eksempel, og uten begrensning, omfatter kommunikasjonsmedier kabelbaserte medier så som kablede nettverk eller en direkte ledningsforbindelse, samt trådløse medier så som akustiske, RF-baserte, infrarøde eller andre trådløse medier. Betegnelsen datamaskinlesbart medium, som anvendt her, omfatter både lagringsmedier og kommunikasjonsmedier.

Forklarende implementasjon

Figur 2 er et funksjonelt blokkdiagram som illustrerer generelt et utviklingssystem for å implementere én utførelsesform av foreliggende oppfinnelse. Systemet omfatter en formateringsspråk-kompilator 202 og en parser 204. Formateringsspråk-kompilatoren 202 og parseren 204 er programvaremoduler (dvs. programmoduler 106 vist i figur 1) som kan befinne seg i en beregningsanordning, så som beregningsanordningen 100 vist i figur 1. Formateringsspråk-kompilatoren 202 tar inn et formatkodet dokument 206. I én utførelsesform er det formatkodede dokumentet 206 et XML-basert dokument. Kort oppsummert omfatter det formatkodede dokumentet 206, som er illustrert i figurene 4-6 og beskrevet i detalj nedenfor, etiketter (ikke vist) som angir deler av en underklasse-definisjon. Som vil bli beskrevet i detalj senere, angir etikettene at det skal skapes en underklasse og assosierte elementer. Når den møter på

disse etikettene, begynner formateringsspråk-kompilatoren 202 å kommunisere med parseren 204 for å bygge underklassen.

I én utførelsesform kan den funksjonaliteten som tilveiebringes av parseren 204 være tilveiebrakt inne i formateringsspråk-kompilatoren 202. I en annen utførelsesform kan den funksjonaliteten som tilveiebringes av parseren 204 være tilveiebrakt ved avledning av en parser-klasse fra en eksisterende parser-klasse i formateringsspråk-kompilatoren 202. Den avledede parser-klassen kan omfatte funksjon-redefinisjoner for hvert av underklassesymbolene (dvs. etikettene) som er definert ifølge foreliggende oppfinnelse. Kort beskrevet kan funksjon-redefinisjonene, som illustrert i figur 10 og beskrevet i detalj nedenfor, være en del av en serie av tilbakekallsfunksjoner som signaliserer en begynnelse av og en slutt på definisjonene av de elementene som er assosiert med underklassen.

Parseren 204 er innrettet for å analysere underklasse-definisjoner innenfor det formatkodede dokumentet 206. Kort beskrevet kompilerer formateringsspråk-kompilatoren 202 innhold inne i det formatkodede dokumentet 206. I én utførelsesform konverterer formateringsspråk-kompilatoren 202 innholdet til en symbolbasert binærstrøm som blir lagret i en symbolbasert binærfil 208. Den symbolbaserte binærfilen 208 kan være på ett av flere formater som er kjente for fagmannen. Den symbolbaserte binærfilen 208 representerer et tre av komponenter som er spesifisert i det formatkodede dokumentet 206. Formateringsspråk-kompilatoren 202 kan imidlertid være uten mulighet for å konvertere noe av innholdet direkte, og dette innholdet kan bli sendt til parseren 206. Underklasse-definisjonene som er definert i det formatkodede dokumentet 206 i henhold til foreliggende oppfinnelse er et eksempel på slikt innhold. Parseren 204 identifiserer egenskaper, hendelser og liknende i underklasse-definisjonene og videresender relevant informasjon om disse elementene til formateringsspråk-kompilatoren 202.

Når den mottar den relevante informasjonen, legger formateringsspråk-kompilatoren 202 til symboler i det formatkodede dokumentet som er assosiert med underklasse-definisjonen. Formateringsspråk-kompilatoren 202 kan også generere representativ kildekode 212 fra hvilken det blir opprettet IL-assembleringer 210. IL-assembleringene 210 omfatter datamaskininstruksjoner for under-

klasser (f.eks. komponenter) som er definert i det formatkodede dokumentet 206. Tidligere ble disse IL-assembleringene generert ved anvendelse av et programvareutviklingsverktøy som kompilerte og linket kildekode skrevet i et programmeringsspråk. Fagmannen vil også forstå at, i en annen utførelsesform, formateringsspråk-kompilatoren 202 kan generere IL-assembleringene 210 uten å generere den symbolbaserte binærfilen 208.

IL-assembleringene 210 som blir opprettet av formateringsspråk-kompilatoren 202 kan gjenbrukes av tradisjonelle programvareutviklingsverktøy. I tillegg kan IL-assembleringene 210 gjenbrukes i andre formatkodede dokumenter. Muligheten for å gjenbruke IL-assembleringene 210 i et formatkodet dokument er illustrert i figur 7 og beskrevet i forbindelse med denne. Foreliggende oppfinnelse gir således komponentutviklere mulighet for på en enkel måte å bygge og utvide komponenter ved anvendelse av et formatteringsspråk. Når nye komponenter har blitt bygget i henhold til foreliggende oppfinnelse, ser de nye komponentene ut som om de er bygget ved anvendelse av et tradisjonelt programmeringsspråk. Utviklere som anvender mekanismen og fremgangsmåten ifølge foreliggende oppfinnelse kan således bygge komponenter uten å lære seg syntaksen og nyansene til ett eller flere programmeringsspråk.

Figur 3 er et funksjonelt blokkdiagram som illustrerer generelt et kjøremiljø for å implementere én utførelsesform av foreliggende oppfinnelse. Kjøremiljøet omfatter en kjøremotor 302, en symbolbasert binærleser 304, og en symbolbasert binærlaster 306. Når kjøremotoren 302 mottar en forespørsel om å laste inn en gitt ressurs (f.eks. det formatkodede dokumentet 206 vist i figur 2), aksesserer kjøremotoren 302 en sideavbildningstabell 308. Sideavbildningstabellen 308 identifiserer hvorvidt det formatkodede dokumentet 206 har en kompilert versjon (f.eks. en symbolbasert binærfil 208). Dersom det eksisterer en kompilert versjon, kommuniserer kjøremotoren 302 med den symbolbaserte binærlasteren 306 for å laste inn den symbolbaserte binærfilen 208. I én utførelsesform identifiserer den symbolbaserte binærfilen 208 IL-assembleringer (f.eks. IL-assembleringen 210) som er assosiert med den symbolbaserte binærfilen 208. Den symbolbaserte binærlasteren 306 laster da de identifiserte IL-assembleringene 210. Når en andel av eller hele den symbolbaserte binærfilen 208 og de assosierte IL-assembleringene 210 har blitt lastet, begynner den

symbolbaserte binærleseren 304 å lese den symbolbaserte binærfilen 208 og IL-assembleringene 210 for å generere interne instruksjoner som blir eksekvert av en prosessor (f.eks. prosesseringsenheten 102 vist i figur 1). Den symbolbaserte binærleseren 304 kan aksessere metadata 310 for å bestemme informasjon så som typer, metoder og hendelser. Generelt omfatter metadataene 310 informasjon om metoder, felter, egenskaper og hendelser. Hvert av disse elementene kan ha sine egne metadata som kan leses for ytterligere detaljer. Ved å anvende metadataene 310, tar således den symbolbaserte binærleseren 304 avgjørelser ved kjøretid for programmatisk å bestemme informasjon om elementene i den symbolbaserte binærfilen 208. I tillegg, som vil bli beskrevet i detalj senere, kan underklasser som originalt er definert i det formatkodede dokumentet 206 bli eksekvert direkte ved anvendelse av IL-assembleringene 210 opprettet i henhold til foreliggende oppfinnelse.

Figurene 4-6 illustrerer en serie av hoveddeler av dokumentkode i det formatkodede dokumentet 206 som illustrerer et eksempel på syntaks for deklarativ definering av underklasser ifølge én utførelsesform av foreliggende oppfinnelse. Figur 10 er en eksempelvis kildekodelisting som illustrerer motsvarende kildekode som formateringsspråk-kompilatoren 202 kan generere basert på dokumentkoden vist i figurene 4-6. I én utførelsesform kan formateringsspråk-kompilatoren 202 generere en fil som inneholder den representative kildekoden. Dette kan for eksempel skje når en utvikler har satt et debug-flagg. Filen som inneholder den representative kildekoden gjør det da mulig for utvikleren å identifisere mulige problemer i teksten i det formatkodede dokumentet 206 eller problemer i formateringsspråk-kompilatoren 202. I en annen utførelsesform trenger ikke den representative koden bli lagret til en fil. I denne utførelsesformen kan formateringsspråk-kompilatoren 202 generere den symbolbaserte binærfilen 208 og IL-assembleringene 210 med eller uten først å generere den første motsvarende kildekoden.

Som en oppsummering beskriver seriene i figurene 4-6 stegvis forskjellige aspekter for deklarativ definering av en underklasse i dokumentkode ifølge foreliggende oppfinnelse. Figur 4 illustrerer et eksempel på syntaks for å definere en underklasse og et underklasse-hierarki. Figur 5 illustrerer et eksempel

på syntaks for å definere identifikatorer, kode samt konstruktorer eller instansieringsmetoder for underklassen. Figur 6 illustrerer et eksempel på syntaks for å definere egenskaper og hendelser for underklassen. Hver av disse figurene vil nå bli beskrevet i detalj.

Figur 4 illustrerer et eksempel på syntaks for å definere en underklasse og et underklasse-hierarki. Den viste delen av dokumentkoden 400 (dvs. underklasse-definisjonen) omfatter en baseklasse-etikett 402 (f.eks. "Button"). For den følgende diskusjonen har hver etikett (f.eks. baseklasse-etiketten) en motsvarende slutt-etikett. For enkelhets skyld er ikke slutt-etiketten referert til i den skrevne beskrivelsen, men er illustrert i de assosierte figurene. Dokumentkoden 400 omfatter en navnerom-attributt 404, en definisjon-navneromdeklarasjon 406 og flere kompilatorhint (f.eks. hintene 408 og 410). Navnerom-attributten 404 identifiserer navnerommet (f.eks. "System.Control") og den assembleringen i hvilken baseklassen (for eksempel Button) er lokalisert. Definisjon-navneromdeklarasjonen 406 angir at enhver attributt innenfor dokumentkoden 400 som inneholder prefikset "def:" representerer et hint til kompilatoren vedrørende handlinger kompilatoren skal utføre.

For eksempel angir hintet 408 (i det følgende betegnet språkhintet 408) at kompilatoren skal generere IL-assembleringen 210 ved anvendelse av det språket (f.eks. C#) som er tilordnet språkhintet 408. Språkhintet 408 kan tilordnes et hvilket som helst blant en rekke mulige programmeringsspråk, så som C, C++ og liknende. Hintet 410 (i det følgende betegnet def:Class-hintet 410) angir for kompilatoren at den skal definere underklassen ved anvendelse et navn 414 (f.eks. MyButton) som er tilordnet def:Class-hintet 410. Hintet def:Class 410 kan også omfatte et underklasse-navnerom 412 som identifiserer det navnerommet (f.eks. "MyControlLib") med hvilket den nye underklassen skal være assosiert. I figur 4 har således en utvikler deklarert en ny underklasse med navn "MyButton" som er avledet fra klassen "Button" som er lokalisert i navnerommet "System.Controls". Den nye underklassen vil bli assosiert med navnerommet "MyControlLib".

I den dokumentkoden 400 som er illustrert i figur 4 er det definert en elementdeklarasjonsetikett 420 (f.eks. "Image"). Fordi et spesifikt navnerom ikke er definert i elementdeklarasjonsetiketten 420, definerer navnerom-attributten 404

også lokaliseringen av det elementet (f.eks. "Image") som er assosiert med elementdeklarasjonsetiketten 420. Elementdeklarasjonsetiketten 420 omfatter et element 421 og en kilde-attributt 422. Kilde-attributten 422 omfatter en egenskap 424 (f.eks. "Source") og en verdi 426 (f.eks. "Happyface.jpg"). Fordi elementdeklarasjonsetiketten 420 befinner seg innenfor underklasse-definisjonen for underklassen definert som "MyButton", vil elementene assosiert med elementdeklarasjonsetiketten 420 bli instansiert når underklassen blir instansiert. I tillegg er elementene som er assosiert med elementdeklarasjonsetiketten 420 inneholdt i en underelement-samling i den nye underklassen. Med andre ord er underklassen moderelementet for elementene som er assosiert med elementdeklarasjonsetikettene 420. Fagmannen vil derfor forstå at dokumentkoden 400 gjør det mulig for utviklere å uttrykke hierarki på en måte som gjør at kompilatoren kan generere elementtrær som har røtter fra underklassen som blir definert (f.eks. "MyButton").

Figur 5 illustrerer et eksempel på syntaks for å definere identifikatorer, kode og konstruktorer for underklassen. Den viste delen av dokumentkoden 500 omfatter dokumentkoden 400 beskrevet ovenfor i tillegg til dokumentkode for å definere identifikatorer, kode, og konstruktorer. For oversiktens skyld er ikke de referansenumrene som er illustrert i figur 4 og beskrevet ovenfor vist i figur 5 dersom de ikke er nyttige for å beskrive de nye aspektene. I figur 5 omfatter elementdeklarasjonsetiketten 420 videre attributter, så som id-attributten 520 og hendelse-attributten 526. Id-attributten 520 omfatter en id-egenskap (f.eks. "ID") og en id-verdi 523 (f.eks. "img1"). Id-attributten 520 illustrerer et eksempel på mekanisme for å definere identifikatorer for underklassen ifølge foreliggende oppfinnelse.

Hendelse-attributten 526 omfatter en hendelse-trigger (f.eks. "DataLoaded") og en hendelse-verdi 529 (f.eks. "OnLoaded"). Hendelse-triggeren 527 spesifiserer hvilken hendelse som skal monitoreres og hendelse-verdien 529 spesifiserer den metoden som eksekveres når hendelse-triggeren 527 aktiveres. Hendelse-verdien 529 er assosiert med en metode 530 (f.eks. funksjonen "OnLoad"). Metoden 530 kan skrives ved anvendelse av et programmeringsspråk. Metoden 530 kan referere til instanser av en klasse og underklasser som er definert i dokumentkoden 500. Når metoden 530 blir skrevet ved an-

vendelse av et programmeringsspråk i et formatkodet dokument, assosieres metoden 530 med et kodehint 540. Kodehintet 540 gjør det mulig for utviklere å legge til kodesnutter i legemet av underklasse-definisjonen. I én utførelsesformetterfølger koden kodehintet 540 og er en kallbar metode eller en hendelsehåndterer. I dokumentkoden 500, for eksempel, tjener funksjonen OnLoaded 530 som hendelsehåndterer for hendelsen DataLoaded som reises av Image-kontrollen. Andre kodesnutter kan også bli lagt til i legemet av underklasse-definisjonen, så som funksjonen CustomInit 550 vist i figur 5.

Dokumentkodden 500 omfatter også et konstruktorhint 542. Konstruktorhintet 542 gjør det mulig for utvikleren å skrive en supplerende konstruktor som supplerer standard-konstruktoren som blir opprettet for en underklasse. I én utførelsesform er den supplerende konstruktoren den siste instruksjonen som eksekveres av den genererte standard-konstruktoren. Den supplerende-konstruktoren kan inneholde kode som påvirker oppførselen til underklassen når den instansieres. Utvikleren kan kalle konstruktoren i underklassen til baseklassen fra den supplerende konstruktoren. Den supplerende konstruktoren vist i figur 5 kaller funksjonen CustomInit 550, som er en brukerdefinert, privat metode.

Figur 6 illustrerer et eksempel på syntaks for å definere egenskaper og hendelser for underklassen. Den viste delen av dokumentkoden 600 omfatter dokumentkoden 400 og dokumentkoden 500 beskrevet ovenfor i tillegg til dokumentkode for å definere egenskaper og hendelser. For oversiktens skyld er ikke de referansenumrene som er illustrert i figurene 4-5 og beskrevet ovenfor vist i figur 6 dersom de ikke er nyttige for å beskrive de nye aspektene.

Dokumentkoden 600 et omfatter egenskap-hint 610 som gjør det mulig å definere egenskaper for underklassen. Egenskapen tjener som en medlemsvariabel i underklassen. Egenskap-hintet 610 kan omfatte én eller flere attributter, så som navn-attributt 612, type-attributt 614, standardverdi-attributt 616 og flagg-attributt 618. Navn-attributten 612 spesifiserer et navn for egenskapen. I én utførelsesform er navnene case-sensitive, og kjøremotoren legger ingen begrensninger på de tegn som kan anvendes i navnet. Navnet må være unikt for den eieren som registrerer det. Type-attributten 614 spesifiserer en type for verdien til egenskapen. Typen omfatter intrinsic, user-defined, struct, class, inter-

face, enum og liknende Standardverdi-attributten 616 spesifiserer en verdi som blir tilordnet når egenskapen blir instansiert. Flagg-attributten 618 spesifiserer typen metoder som opprettes av wrapperen når egenskapen blir instansiert. Flaggene kan styre karakteristikkene ved egenskapen, så som ReadOnly, Inheritable to child elements, Private og liknende.

Dokumentkoden 600 omfatter også et hendelse-hint 620. Hendelse-hintet 620 gjør det mulig å definere hendelser for underklassen. Hendelse-hintet 620 kan omfatte attributter, så om navn-attributt 622, route-attributt 624, flagg-attributt 626 og liknende. Navn-attributten 622 spesifiserer den strengen som refererer til denne hendelsen. Når en xml-fil anvender underklassen, anvender xml-filen denne strengen for å binde den korrekte applikasjonsutvikler-definerte koden. Route-attributten 624 spesifiserer den metoden som bestemmer hvilke elementer i treet hendelsen skal trigges for. Flagg-attributten 626 spesifiserer andre karakteristikker som er assosiert med hendelsen. For eksempel, for dokumentkoden 600, vil formatteringsspråk-kompilatoren generere kode for å aktivere en hendelse kalt DblClick og inkludere informasjon assosiert med hendelsen, så som den støttede rutingen, flagg og liknende. Fagmannen vil forstå at navnene og verdiene assosiert med attributtene kan modifiseres innenfor rammen til foreliggende oppfinnelse.

Dokumentkoden 600 omfatter også en hendelsehåndterer-deklarasjon 630. Hendelsehåndterer-deklarasjonen 630 kan ha assosierte hendelsehåndterer-modifikatorer 632, så som public/private, delegate, returtype og liknende. I figur 6 deklarerer hendelsehåndterer-deklarasjonen 630 en hendelsehåndterer (dvs. metode) (f.eks. "DblClickEventHandler") som blir kalt når den assosierte hendelsen inntreffer.

Andre etiketter kan også defineres i formatkodede dokumenter. For eksempel kan en rot-etikett (f.eks. "def:Library") bli anvendt for å informere kompilatoren og/eller parseren om å definere underklassen i en separat fil, for å definere underklassen med andre underklasser i én fil og liknende. Det kan deklareres et navnerom for et spesialanpasset element, for alle klasser som blir definert innenfor i rot-etiketten og liknende.

Figur 7 illustrerer en hoveddel av et formatkodet dokument som illustrerer et eksempel på syntaks for bruk av en underklasse fra et formatkodet dokument. Dokumentkoden 700 omfatter et rot-element 702 (f.eks. et FlowPanel element), en standardnavnerom-deklarasjon 704, et definisjon-navnerom prefiks 706 og et element 708. En vil forstå at elementet 708 kan referere til en spesialkonstruert komponent som har blitt bygget ved anvendelse av den ovenfor beskrevne syntaksen. Standardnavnerom-deklarasjonen 704 spesifiserer et standard navnerom for etiketter uten prefiks. I den illustrerte dokumentkoden 700 referer standardnavnerom-deklarasjonen 704 til navnerommet "System.Controls". Definisjon-navnerom prefikset 706 spesifiserer en lokasjon der den spesialkonstruerte komponenten eller elementet kan bli funnet. I den illustrerte dokumentkoden 700 refererer definisjon-navnerom prefikset 706 til "MyControlLib". Elementet 708 omfatter et klassenavn 710 (f.eks. "MyControlLib"), en identifikator 712 og en tekststreng 714. Klassenavnet 710 refererer til den klassen som blir instansiert for dette elementet. Identifikatoren 712 refererer til elementet og omfatter et navn (f.eks. "ID") og en verdi (f.eks. "button1"). Verdien blir navnet på instansen av klassenavnet 710 ved eksekvering (f.eks. button1).

Generalisert operasjon av den forklarende implementasjonen

Figur 8 er et logisk flytdiagram som illustrerer generelt en prosess 800 for å kompilere deklarativt definerte underklasser ifølge én utførelsesform av foreliggende oppfinnelse. Den eksempelvise dokumentkoden som er vist i figurene 4-6, sammen med den motsvarende kildekoden i figur 10, blir anvendt i forbindelse med figur 8 for å beskrive prosessen 800. Som en lett innser, etter å ha sammenliknet figur 10 med dokumentkoden 800, gjør foreliggende oppfinnelse det mulig å generere sofistikerte underklasser på en forklarende måte uten å kreve at utvikleren forstår det underliggende programmeringsspråket. Den representative koden i figur 10 er C# kildekode. Formateringsspråk-kompilatoren, ifølge foreliggende oppfinnelse, kan imidlertid generere representativ kode ved anvendelse av syntaksen til et hvilket som helst programmeringsspråk.

Prosessen 800 begynner i blokk 801, der en formateringsspråk-kompilator kompilerer et formatkodet dokument og møter dokumentkode som omfatter en definisjon av en underklasse. Formateringsspråk-kompilatoren kan avgjøre at den har møtt en underklasse-definisjon ved hjelp av en hvilken som helst mekanisme, så som at den ikke gjenkjenner dokumentkoden som noe annet format, at den kjenner igjen en underklasse-etikett (f.eks. def:Class), og liknende. For eksempel, i dokumentkoden 400, kan formateringsspråk-kompilatoren prosessere flere setninger før den møter den underklasse-etiketten (f.eks. def:Class hintet 410) som identifiserer underklasse-definisjonen. I andre utførelsesformer kan underklasse-etiketten møtes før andre setninger, for eksempel når underklassen ikke spesifiserer en baseklasse (f.eks. button). Generelt kan underklasse-etiketten dukke opp hvor som helst der en element-etikett kan dukke opp i dokumentkode. Når dokumentkoden detekteres å omfatte en underklasse-definisjon, fortsetter prosesseringen i blokk 802.

I blokk 802 blir underklasse-etiketten prosessert. Som beskrevet i forbindelse med figur 4, blir underklasse-etiketten (også referert til som def:Class hintet 410) tilordnet et navn 414, som også kan omfatte et underklasse-navnerom 412. Basert på denne informasjonen kan det bli generert motsvarende kode, så som linje 1 i figur 10, for assembleringen. I tillegg blir det generert en klasse som har det tilordnede navnet, så som vist i linje 5 i figur 10. Linje 5 vil bli utvidet med annen generert kode straks ytterligere setninger (f.eks. navnerom og baseklasse) er prosessert. Som en del av prosesseringen av underklasse-etiketten, identifiserer prosessen hvorvidt underklassen er avledet fra noen klasse (dvs. en baseklasse), og innhenter informasjon assosiert med baseklassen. I figur 4 er underklassen som blir opprettet, "MyButton", avledet fra "Button", som definert via baseklasse-etiketten 402. Linje 5 i figur 10 omfatter således motsvarende kode som avleder MyButton fra Button som vist.

I tillegg blir det generert en standard konstruktor for underklassen. Igjen, ved prosessering av ytterligere setninger, kan denne standard konstruktoren bli utvidet med ytterligere representativ kode. Linjene 23-24 og 26 i figur 10 svarer

til den representative koden som blir generert. "this.__Initialize_This()" er imidlertid kode som blir lagt til etter prosessering av en annen setning, som vil bli beskrevet nedenfor. Når underklassen er prosessert, fortsetter prosesseringen i blokk 804.

I blokk 804 hentes den neste setningen i dokumentkoden. Den neste setningen kan være setningen før eller kan være setningen etter underklasseetikett-setningen avhengig av lokaliseringen av underklasseetikett-setningen innenfor underklasse-definisjonen. Når den neste setningen er lest inn, fortsetter prosesseringen til bestemmelsesblokken 806.

Ved bestemmelsesblokken 806 avgjøres det hvorvidt setningen definerer en klasse. Som nevnt ovenfor, gjør foreliggende oppfinnelse det mulig å uttrykke en klasse på en forklarende måte. I én utførelsesform vil en setning som definerer en annen klasse (dvs. et element) som er en avkommer av underklassen forekomme etter underklasseetikett-setningen. For nå, antatt at den aktuelle setningen ikke definerer en klasse, fortsetter prosesseringen til blokk 808.

I blokk 808 blir setningen prosessert. Det finnes forskjellige typer setninger som kan forekomme innenfor underklasse-definisjonen. Disse setningene kan forekomme i forskjellig rekkefølge. Prosesseringen involvert for hver type setning vil bli beskrevet nedenfor i detalj. Generelt imidlertid, vil hver setning som blir prosessert resultere i generering av ytterligere motsvarende kode for underklasse-definisjonen. Når én av setningene har blitt prosessert i blokk 808, fortsetter prosesseringen til bestemmelsesblokk 810.

Ved bestemmelsesblokk 810 avgjøres det hvorvidt det er flere setninger som skal prosesseres innenfor underklasse-definisjonen. Dersom det finnes en annen setning, går prosessen tilbake til blokkene 804-808 og prosesserer den setningen. Når alle setningene innenfor underklasse-definisjonen er prosessert, fortsetter prosessen fra bestemmelsesblokk 810 til blokk 812.

I blokk 812 blir den representative koden som har blitt generert under den ovennevnte prosessen anvendt for å generere IL-assembleringene for underklasse-definisjonen. Utvikleren kan ha spesifisert i én av kompilatorhint-setningene (blokk 820) det programmatiske språket som skal anvendes ved generering av den representative koden, kan ha spesifisert i hvilken assembler-fil underklassen skal lagres, og liknende. Når IL-assembleringen eller IL-assem-

bleringene har blitt generert, er prosesseringen komplett. Som nevnt ovenfor er den ovenfor genererte assembler-filen nå tilgjengelig for å bli eksekvert ved anvendelse av tradisjonelle programmeringsspråk, ved anvendelse av formatkodede setninger ifølge foreliggende oppfinnelse, og liknende. Assembler-filen fremtrer som om den skulle være skrevet ved anvendelse av et tradisjonelt programmatisk språk.

Tilbake til bestemmelsesblokk 806, dersom setningen begynner en definisjon av en ny klasse (dvs. et element), fortsetter prosesseringen i blokk 814. I blokk 814 blir setningene som hører til denne nye klassen prosessert ved anvendelse av prosessering i blokk 808 for hver setning inntil alle setninger som hører til den nye klassen er prosessert. For eksempel, i figur 5, blir setningene 520, 526, 422 prosessert for den nye klassen "Image". Image er da en avkommer fra underklassen "MyButton". Prosessen fortsetter i blokk 804 for å fortsette prosessering av setninger assosiert med underklassen.

Nå, tilbake til blokk 808, skal prosesseringen for hver individuelle type setning (blokkene 820-832) som kan forekomme innenfor underklasse-definisjonen beskrives. I blokk 820 blir setninger som er kompilatorhint prosessert. Generelt tilveiebringer de setningene som er kompilatorhint informasjon til formateringsspråk-kompilatoren vedrørende hvordan assembler-filen skal genereres. For eksempel, i figur 4, er språkhintet 408 et kompilatorhint som informerer formatteringsspråk-kompilatoren om hvilket programmeringsspråk den skal anvende for å generere den motsvarende koden. I figur 4 er språkhintet 408 satt til C#, og den representative koden illustrert i figur 10 er derfor C# kildekode.

I blokk 822 blir setninger som definerer navnerom prosessert. Disse navnerommene blir da inkludert i den representative koden, så som vist i linjene 2 og 3 i figur 10.

I blokk 824 blir setninger som definerer en identifikator (id) for en klasse prosessert. For enhver id-attributt i det formatkodede dokumentet genererer formatteringsspråk-kompilatoren en medlemsvariabel i den assosierte klassen. Medlemsvariabelen blir generert med en type som er den samme som den klassen for hvilken id-attributten er definert. Medlemsvariabelens navn er den id-verdien som er tilordnet id-attributten. For eksempel, i figur 5, under prosessering av de setningene som hører til klassen Image, møtes en id-attributt 520.

Som vist i linje 7 i figur 10, vil således formateringsspråk-kompilatoren generere representativ kode for klassen MyButton som har en medlemsvariabel definert som:

```
private System.Comtrols.Image img1;
```

Ved kjøretid blir medlemsvariabelen initialisert til en instans av den motsvarende typen (f.eks. Image) som blir opprettet i metoden InitComponent(). Som resultat av dette kan enhver kode i klassen MyButton aksessere enhver annen elementinstans i sitt hierarki bare med klassens id-verdi. Initialiseringen av medlemsvariabelen er vist i linje 37 i figur 10.

I blokk 826 blir setninger som definerer en egenskap for en klasse prosessert. For enhver egenskap som er definert i det formatkodede dokumentet, genererer formateringsspråk-kompilatoren motsvarende kildekode for egenskapen og registrerer egenskapen dersom det er nødvendig. For eksempel, i figur 6, inkluderer dokumentkoden 600 et egenskap-hint 610 som har forskjellige attributter 612-618. Egenskap-hintet 610 vil informere formateringsspråk-kompilatoren om at setningene som følger etter egenskap-hintet 610 er for en egenskap. Attributene 612-618 blir da lest inn som egenskap-setninger. Formateringsspråk-kompilatoren vil generere motsvarende kode for "Label"-egenskapen for å registrere egenskapen for klassen MyButton som vist i linjene 9-12 i figur 10. Formateringsspråk-kompilatoren vil også generere motsvarende kode for å definere egenskapen som vist i linjene 28-30 i figur 10. Linjene 28-30 illustrerer hvordan navn-attributten 612 blir egenskapens navn i den motsvarende koden og type-attributten blir egenskapens type i den motsvarende koden.

I én utførelsesform genererer prosessen kode for egenskapens verdier ved å kalle TypeConverter for egenskapens type, som eksekverer kode ved kjøretid ved anvendelse av refleksjon og konverterer strengen til en faktisk objektinstans av egenskapens type. I en annen utførelsesform kaller prosessen 800 type-konvertereren for egenskapen for å oppnå den faktiske objektverdien og konverterer verdien til et InstanceDescriptor-objekt. InstanceDescriptor-objektet inneholder tilstrekkelig informasjon, slik at formateringsspråk-kompilatoren basert på objektet kan generere representativ kode som oppretter en ny instans av

verdiens type som spesifisert i attributtene assosiert med egenskap-hintet. Linjene 28-30 i figur 19 illustrerer hvordan labelProperty vil bli satt for en instans av MyButton under kjøretid med den tilordnede verdien for Label-egenskapen. Den tilordnede verdien kan være tilordnet på forklarende måte i dokumentkode (som vist i figur 7) eller tilordnet programmatisk ved anvendelse av et hvilket som helst programmeringsspråk.

I blokk 828 blir setninger som definerer en hendelse for en klasse prosessert. For hendelse-setninger genererer formateringsspråk-kompilatoren motsvarende kode for hendelsen. Hendelser kan bli definert i dokumentkode ved anvendelse av forskjellige mulige mekanismer, så som en hendelse-attributt 526 vist i figur 5 og som et hendelse-hint 620 vist i figur 6. Først vil mekanismen som anvender hendelse-attributten bli beskrevet. Hendelse-attributten omfatter hendelse-triggeren og hendelse-verdien. Hendelse-triggeren svarer til hendelsen og hendelse-verdien svarer til den funksjonen som blir eksekvert når hendelse-triggeren aktiveres. Med henvisning til figur 5 blir hendelse-verdien 529 ("OnLoaded") lagt til som en hendelsehåndterer i linjene 38-40 i den motsvarende koden i figur 10 ved å definere "this.Onloaded" som den nye System.Controls.DataLoadedEventHandler. Hendelse-triggeren 527 ("DataLoaded") blir lagt til som hendelsen i linje 38 i den motsvarende koden i figur 10 ved å definere den første parameteren til AddHandler som System.Controls.Image.DataLoadedEvent. Den andre parameteren for AddHandler kan anvende standardverdier eller kan være spesifisert i dokumentkoden.

Mekanismen som anvender hendelse-hintet 620 vil nå bli beskrevet. Hendelse-hintet 620 omfatter en hendelsenavn-attributt og andre attributter. Navnet som er tilordnet hendelsenavn-attributten blir registrert som hendelsen. For eksempel, med henvisning til figur 6, er "DblClick" navnet som er tilordnet hendelsenavn-attributten. "DblClick" er derfor den første parameteren i metoden RegisterEvent som blir generert i linjene 14-16 i figur 10. Typen til hendelsehåndterer-deklarasjonen 630 for metoden "DblClickEventHdlr" er også inkludert som en parameter i metoden RegisterEvent i figur 10. Igjen kan de andre parameterne til metoden RegisterEvent anvende standardverdier eller kan være spesifisert i det formatkodede dokumentet ved anvendelse av andre attributter.

Ved kjøretid, ifølge én utførelsesform, blir hendelsen innkoplet ved anvendelse av refleksjon for å identifisere typen av DblClickEventHandler og for å lokalisere metoden.

I blokk 830 blir setninger som definerer kode prosessert. Setninger som definerer kode etterfølger en kompilatorhint-setning (blokk 820), så som kodehint. Disse setningene er skrevet i et programmeringsspråk og vil bli inkludert som de er i den motsvarende koden. For eksempel illustrerer figur 5 metoden 530 og funksjonen CustomInit 550. Metoden 530 og funksjonen CustomInit 550 befinner seg henholdsvis i linjene 18-19 og 21 i figur 10.

I blokk 832 blir setninger som definerer en supplerende konstruktor prosessert. Igjen etterfølger setninger som definerer en supplerende konstruktor en kompilatorhint-setning (blokk 820), så som konstruktør-hint. Formateringsspråk-kompilatoren genererer en supplerende konstruktor ("this.__Initialize_This()" i figur 10) og legger til setningene i den supplerende konstruktoren. For eksempel, i figur 5, kaller setningen etter konstruktør-hintet 542 "CustomInit()". I figur 10 i linje 33 legger således formateringsspråk-kompilatoren til "CustomInit() i den supplerende konstruktoren i den motsvarende kildekoden.

Fagmannen vil forstå at blokk 812 kan bli utført stegvis under prosessering i boksene 804-810 uten å avgå fra foreliggende oppfinnelse. I tillegg, avhengig av den funksjonelle implementasjonen av parser 204 og formateringsspråk-kompilator 202, kan det være tilbakekall mellom parser 204 og formateringsspråk-kompilator 202 under prosessen 800.

Figur 9 er et logisk flytdiagram som illustrerer generelt en kjøreprosess 900 for å anvende en underklasse som har blitt deklarert fra inne i et formatkodet dokument ifølge én utførelsesform av foreliggende oppfinnelse. Dokumentkodeeksempelet som er vist i figur 7 blir anvendt i forbindelse med figur 9 for å beskrive prosessen 900. Prosessen 900 begynner i blokk 901, der en kjøremotor mottar en forespørsel om en gitt ressurs (f.eks. et formatkodet dokument) og bestemmer at det eksisterer en kompilert versjon av den gitte ressursen. For eksempel, i figur 7, antatt at dokumentkoden 700 ikke allerede har blitt kompilert til en symbolbasert binærfil, når formateringsspråk-kompilatoren møter på elementet 708, vil formateringsspråk-kompilatoren detektere at klassen MyButton har en assosiert symbolbasert binærfil og IL-assemblering. Denne assosierte,

symbolbaserte binærfilen som inkluderer de symboliserte attributtene for klassen MyButton vil da bli prosessert ved anvendelse av prosessen 900. Den assosierte, symbolbaserte binærfilen og IL-assembleringen kan være generert under anvendelse av foreliggende oppfinnelse. Prosesseringen fortsetter i blokk 902.

I blokk 902 blir den symbolbaserte binærfilen lastet. Den symbolbaserte binærfilen kan bli lastet stegvis eller i sin helhet. Den symbolbaserte binærfilen kan identifisere IL-assembleringer assosiert med den symbolbaserte binærfilen som må lastes. Med henvisning til figur 7, for eksempel, blir de IL-assembleringene som er assosiert med klassen MyButton lastet. Prosesseringen fortsetter i blokk 904.

I blokk 904 blir et symbol hentet fra den symbolbaserte binærfilen. Som nevnt ovenfor, i det nye kjøremiljøet, er den symbolbaserte binærfilen som blir generert uavhengig av hvilket programmeringsspråk som ble anvendt for å generere den symbolbaserte binærfilen. Kjøremotoren kan således prosessere den symbolbaserte binærfilen uten hensyn til hvilket programmeringsspråk som opprinnelig ble anvendt for å skape den symbolbaserte binærfilen. Prosesseringen fortsetter ved bestemmelsesblokk 906.

Ved bestemmelsesblokk 906 avgjøres det hvorvidt symbolet som ble innhentet er en hendelse. Dersom symbolet ikke er en hendelse, fortsetter prosesseringen i blokk 908.

I blokk 908 blir symbolet prosessert. Som nevnt ovenfor avhenger ikke prosesseringen av symbolet av måten på hvilken den symbolbaserte binærfilen har blitt generert. Med andre ord vil symbolprosesseringen av et symbol i en symbolbasert binærfil som har blitt opprettet på en forklarende måte i et formatkodet dokument eller ved anvendelse av et programmeringsspråk fortsette på samme måte. Derfor, ettersom prosessering av symboler i en symbolbasert binærfil kjent for fagmannen, trenger ikke prosesseringen av symbolet å bli beskrevet utførlig her. Prosesseringen fortsetter ved bestemmelsesblokk 910.

Ved bestemmelsesblokk 910 avgjøres det hvorvidt det er flere symboler i den symbolbaserte binærfilen. Dersom det er flere symboler, går prosessen tilbake til blokk 904 og fortsetter som beskrevet ovenfor. På den annen side, der-

som det ikke finnes flere symboler, er prosesseringen fullført og fortsetter til slutt-blokken.

Tilbake til bestemmelsesblokk 906, dersom symbolet er en hendelse, fortsetter prosesseringen i blokk 912. I blokk 912 blir metadata assosiert med den symbolbaserte binærfilen lastet. Metadataene omfatter beskrivelser av typer, ressurser, metoder og liknende. Prosesseringen fortsetter i blokk 914.

I blokk 914, ved anvendelse av metadata og refleksjon, finner prosessen en mål-elementtype for hendelsen. Dette involverer det å aksessere metadataene og gå gjennom hvert felt for å bestemme typen. Prosesseringen fortsetter i blokk 916.

I blokk 916 blir hendelsen validert med hensyn til mål-elementtypen. Dette sikrer at hendelsen er en gyldig hendelse. Dersom hendelsen ikke er en gyldig hendelse for mål-elementtypen, rapporteres en feil. Prosesseringen fortsetter i blokk 918.

I blokk 918 anvendes refleksjon over mål-elementtypen for å oppnå hendelsemetoden, som deretter blir eksekvert. Det å eksekvere hendelsemetoden omfatter det å eksekvere kode i den assosierte IL-assembleringen. Med henvisning til figur 7, når MyButton blir instansiert og hendelsen er funnet, blir således hendelsemetoden ("DblClickEvent") tilknyttet. Dette knytter en hendelsehåndterer (f.eks. DblClickEventHandler) til hendelsen (f.eks. DblClick). Prosesseringen fortsetter ved bestemmelsesblokk 910 og forløper som beskrevet ovenfor,

Som beskrevet tilveiebringer således foreliggende oppfinnelse en mekanisme for forklarende definering av underklasser i et formatkodet dokument og for anvendelse av underklasser fra et formatkodet dokument. Dette gjør at utviklere kan fokusere mer på måter å anvende komponentene i stedet for å bekymre seg for hvordan de skal implementere komponentene i et programmeringsspråk.

Spesifikasjonen, eksemplene og dataene ovenfor tilveiebringer en komplett beskrivelse av opprettelse og anvendelse av sammensetningen ifølge oppfinnelsen. Ettersom mange utførelsesformer av oppfinnelsen er mulige innenfor idéen bak og rammen til oppfinnelsen, defineres oppfinnelsen av de etterfølgende patentkravene.

PATENTKRAV

1. Datamaskinlesbart medium innkodet med en datamaskinlesbar datastruktur, der datastrukturen omfatter:

et skjema for forklarende definering av en underklasse-definisjon i et formatkodet dokument, idet skjemaet kan bli kompilert til en eksekverbar assemblering som instansierer et objekt som er assosiert med en underklasse som er definert innenfor underklasse-definisjonen ved eksekvering av assembleringen, hvor den eksekverbare assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den eksekverbare sammenstillingen er kompilert uten hensyn til programmeringsspråket.

2. Datamaskinlesbart medium ifølge krav 1, der skjemaet er et XML (Extensible Markup Language) – basert skjema.

3. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter en underklasse-etikett for å definere et navn, idet navnet er assosiert med en type for objektet som blir instansiert.

4. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter et språkhint for å spesifisere et programmeringsspråk å anvende ved kompilering av underklasse-definisjonen.

5. Datamaskinlesbart medium ifølge krav 4, der skjemaet videre omfatter et kodehint for å innlemme kode i det formatkodede dokumentet, idet koden blir kompilert direkte inn i den eksekverbare assembleringen.

6. Datamaskinlesbart medium ifølge krav 5, der koden blir kompilert direkte basert på en syntaks for det programmeringsspråket som er spesifisert i språkhintet.

7. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter et baseklasse-hint for å spesifisere en baseklasse fra hvilken underklassen er avledet.

8. Datamaskinlesbart medium ifølge krav 7, der skjemaet omfatter en baseklassenavnerom-attributt for å spesifisere et navnerom i hvilket baseklassen befinner seg.

9. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter en underklasse-etikett for å spesifisere underklassen.

10. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter et konstruktør-hint for å spesifisere én eller flere supplerende handlinger å utføre ved instansiering av et objekt.

11. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter et hendelse-hint for å spesifisere en hendelse og en tilhørende hendelsehåndterer som assosieres med objektet når det blir instansiert.

12. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter et egenskap-hint for å spesifisere en egenskap, idet egenskapen blir en medlemsvariabel i objektet når objektet blir instansiert.

13. Datamaskinlesbart medium ifølge krav 1, der skjemaet omfatter en elementdeklarasjon for å spesifisere et avkommerelement av underklassen.

14. Datamaskinlesbart medium som inneholder instruksjoner for å generere en eksekverbar assemblering for en underklasse, der instruksjonene omfatter det å:

identifisere en underklasse-definisjon i et formatkodet dokument, idet underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan bli eksekvert for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et

programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

15. Datamaskinlesbart medium ifølge krav 14, der det å identifisere underklasse-definisjonen omfatter det å parsere en underklasse-etikett, der underklasse-etiketten omfatter en underklassenavn-attributt for å definere et navn, idet navnet er assosiert med en type for objektet som blir instansiert.

16. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å kompilere underklasse-definisjonen til motsvarende kildekode som blir kompilert inn i assembleringen.

17. Datamaskinlesbart medium ifølge krav 16, der den motsvarende kildekoden er basert på et programmatisk språk.

18. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å kompilere kode innenfor underklasse-definisjonen inn i assembleringen, idet koden blir innlemmet direkte i underklasse-definisjonen av et kodehint.

19. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere en baseklasse-etikett som definerer en baseklasse fra hvilken underklassen skal avledes, spesifisere arven ved anvendelse av en syntaks assosiert med et programmatisk språk, samt kompilere syntaksen til å bli en del av assembleringen.

20. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et konstruktør-hint som definerer minst én supplerende handling som blir utført når objektet instansieres, idet den supplerende handlingen er en del av assembleringen.

21. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et hendelse-hint som oppretter en hendelse-definisjon og en tilhørende hendelsehåndterer for underklassen.

22. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et egenskap-hint som definerer en egenskap som blir en medlemsvariabel i objektet når objektet blir instansiert.

23. Datamaskinlesbart medium ifølge krav 14, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere en element-deklarasjon-etikett som definerer et avkommerelement av underklassen.

24. Datamaskin-implementert fremgangsmåte for å generere en eksekverbar assemblering, idet fremgangsmåten omfatter det å:

identifisere en underklasse-definisjon i et formatkodet dokument, der underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan eksekveres for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

25. Datamaskin-implementert fremgangsmåte ifølge krav 24, der det å identifisere underklasse-definisjonen omfatter det å parsere en underklasse-etikett, der underklasse-etiketten omfatter en underklassenavn-attributt for å definere et navn, idet navnet assosieres med en type for objektet som instansieres.

26. Datamaskin-implementert fremgangsmåte ifølge krav 24, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere en baseklasse-etikett som definerer en baseklasse fra hvilken underklassen av-

ledes, spesifisere arven ved anvendelse av en syntaks som er basert på et programmatisk språk, samt kompilere syntaksen til å bli en del av assembleringen.

27. Datamaskin-implementert fremgangsmåte ifølge krav 24, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et konstruktør-hint som definerer minst én supplerende handling som utføres når objektet instansieres, idet den supplerende handlingen er en del av assembleringen.

28. Datamaskin-implementert fremgangsmåte ifølge krav 24, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et hendelse-hint som oppretter en hendelse-definisjon og en tilhørende hendelsehåndterer for underklassen.

29. Datamaskin-implementert fremgangsmåte ifølge krav 24, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et egenskap-hint som definerer en egenskap som er en medlemsvariabel i objektet når objektet instansieres.

30. Datamaskinlesbart medium som omfatter instruksjoner for å skape en eksekverbar assemblering for en underklasse, idet instruksjonene omfatter fremgangsmåte ifølge krav 24.

31. Datasystem for å generere en assemblering fra en underklasse-definisjon med et formatkodet dokument, der datasystemet omfatter:

en prosessor; og

et minne, der minnet er allokert for flere datamaskin-eksekverbare instruksjoner som blir lastet inn i minnet, hvor prosessoren eksekverer instruksjonene for å:

identifisere en underklasse-definisjon i et formatkodet dokument, idet underklasse-definisjonen definerer en underklasse; og

generere en assemblering basert på underklasse-definisjonen, idet assembleringen kan bli eksekvert for å opprette en instans av et objekt assosiert

med underklassen, hvor den genererte assemblering er uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

32. Datamaskin-implementert fremgangsmåte ifølge krav 31, der det å identifisere underklasse-definisjonen omfatter det å parsere en underklasse-etikett, der underklasse-etiketten omfatter en underklassenavn-attributt for å definere et navn, idet navnet assosieres med en type for objektet som blir instansiert.

33. Datamaskin-implementert fremgangsmåte ifølge krav 31, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et konstruktør-hint som definerer minst én supplerende handling som utføres når objektet instansieres, idet den supplerende handlingen er en del av assembleringen.

34. Datamaskin-implementert fremgangsmåte ifølge krav 31, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et hendelse-hint som oppretter en hendelse-definisjon og en tilhørende hendelsehåndterer for underklassen.

35. Datamaskin-implementert fremgangsmåte ifølge krav 31, der det å generere en assemblering basert på underklasse-definisjonen omfatter det å parsere et egenskap-hint som definerer en egenskap som er en medlemsvariabel i objektet når objektet blir instansiert.

36. Datamaskin-implementert fremgangsmåte for å generere en eksekverbar assemblering, idet fremgangsmåten omfatter:

et middel for å identifisere en underklasse-definisjon i et formatkodet dokument, der underklasse-definisjonen definerer en underklasse; og

et middel for å generere en assemblering basert på underklasse-definisjonen, der assembleringen kan eksekveres for å opprette en instans av et objekt assosiert med underklassen, hvor den genererte assemblering er

uavhengig av et programmeringsspråk som opprinnelig er benyttet for å skape assembleringen slik at den genererte sammenstillingen er eksekvert uten hensyn til programmeringsspråket.

37. Datamaskin-implementert fremgangsmåte ifølge krav 36, der middelet for å identifisere underklasse-definisjonen omfatter et middel for å parsere en underklasse-etikett, idet underklasse-etiketten omfatter en underklassenavn-attributt for å definere et navn, og navnet assosieres med en type for objektet som blir instansiert.

38. Datamaskin-implementert fremgangsmåte ifølge krav 36, der middelet for å generere en assemblering basert på underklasse-definisjonen omfatter en måte for å parsere en baseklasse-etikett som definerer en baseklasse fra hvilken underklassen avledes, et middel for å spesifisere arven ved anvendelse av en syntaks som er basert på et programmatisk språk, samt en måte for å kompilere syntaksen til å bli en del av assembleringen.

39. Datamaskin-implementert fremgangsmåte ifølge krav 36, der middelet for å generere en assemblering basert på underklasse-definisjonen omfatter et middel for å parsere et konstruktør-hint som definerer minst én supplerende handling som utføres når objektet instansieres, idet den supplerende handlingen er en del av assembleringen.

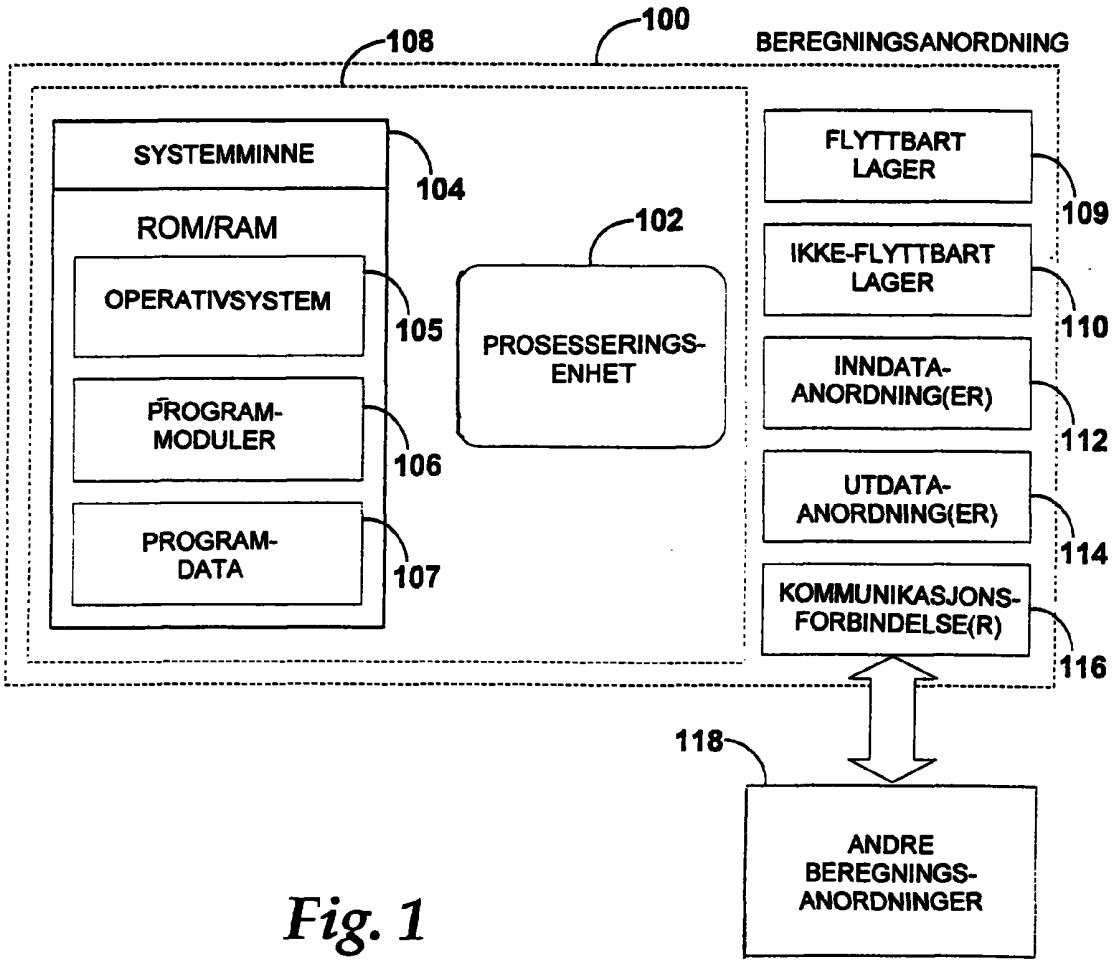

100
BEREGNINGSANORDNING
108
SYSTEMMINNE
104
ROM/RAM
OPERATIVSYSTEM
105
PROGRAM-MODULER
106
PROGRAM-DATA
107
102
PROSESSERINGS-ENHET
FLYTTBART LAGER
109
IKKE-FLYTTBART LAGER
110
INNDATA-ANORDNING(ER)
112
UTDATA-ANORDNING(ER)
114
KOMMUNIKASJONS-FORBINDELSE(R)
116
118
ANDRE BEREGNINGS-ANORDNINGER

Fig. 1

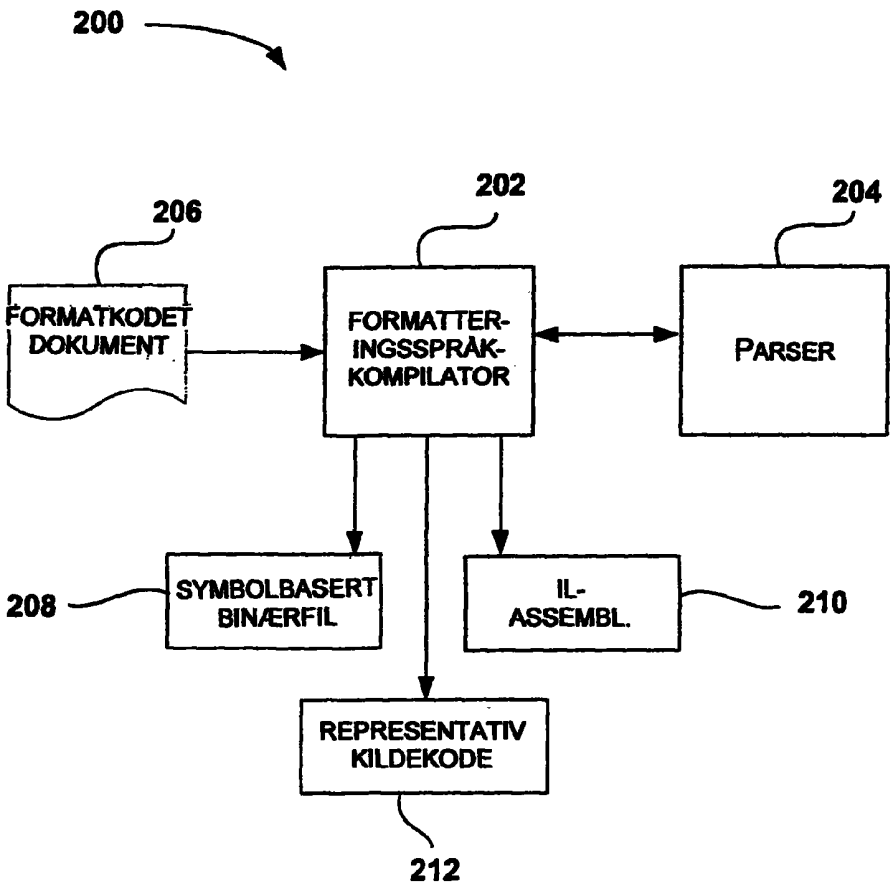

200
206
202
204
FORMATKODET DOKUMENT
FORMATTER-INGSSPRÅK-KOMPILATOR
PARSER
208
SYMBOLBASERT BINÆRFIL
IL-ASSEMBL.
210
REPRESENTATIV KILDEKODE
212

Fig. 2

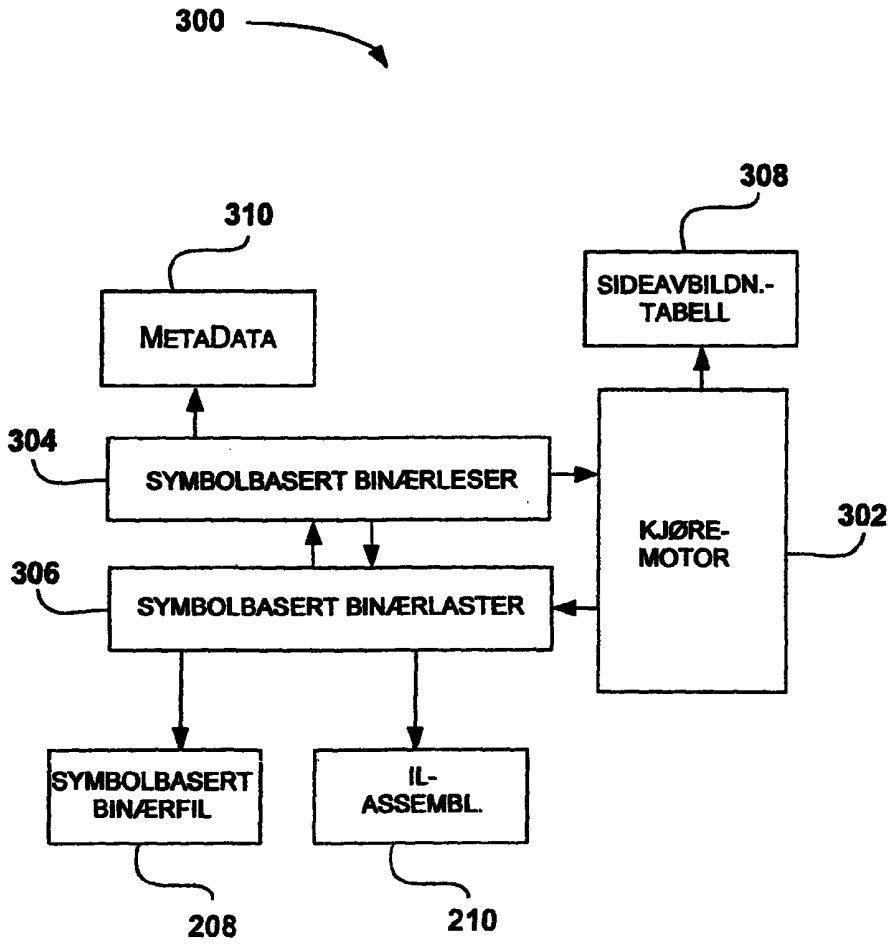

300
308
310
SIDEAVBILDN.-
TABELL
METADATA
304
SYMBOLBASERT BINÆRLESER
KJØRE-
MOTOR
302
306
SYMBOLBASERT BINÆRLASTER
SYMBOLBASERT
BINÆRFIL
IL-
ASSEMBL.
208
210

Fig. 3

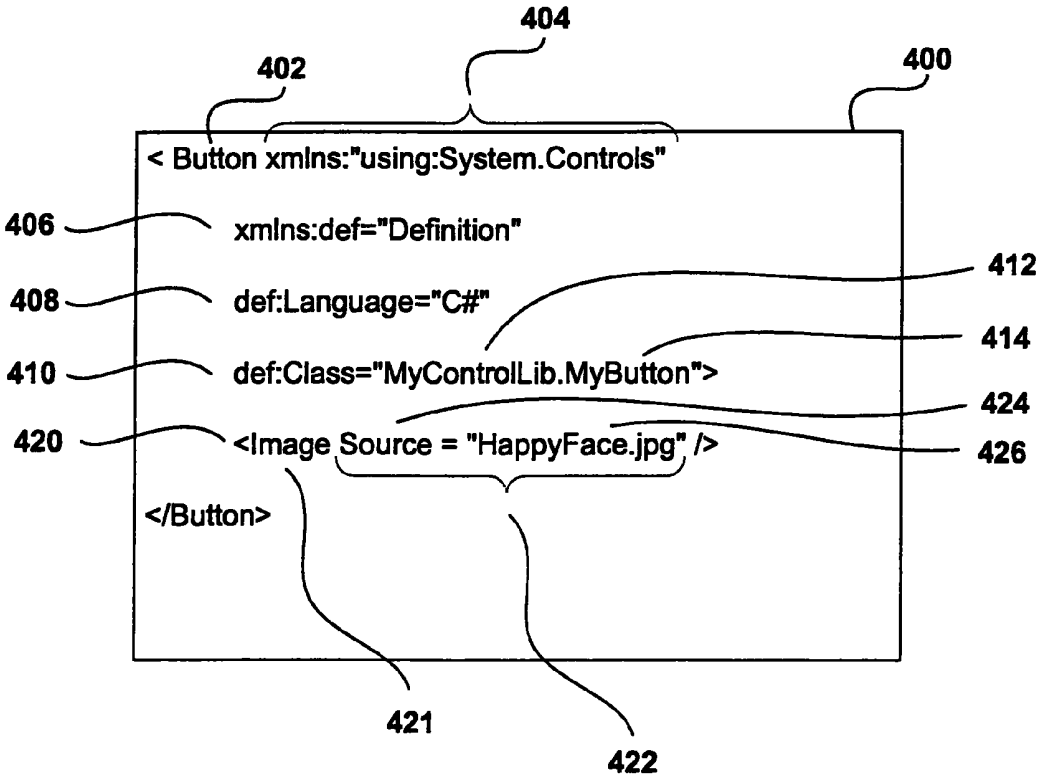

404
402
400
< Button xmlns:"using:System.Controls"
406
xmlns:def="Definition"
412
408
def:Language="C#"
414
410
def:Class="MyControlLib.MyButton">
424
420
<Image Source = "HappyFace.jpg" />
426
</Button>
421
422

Fig. 4

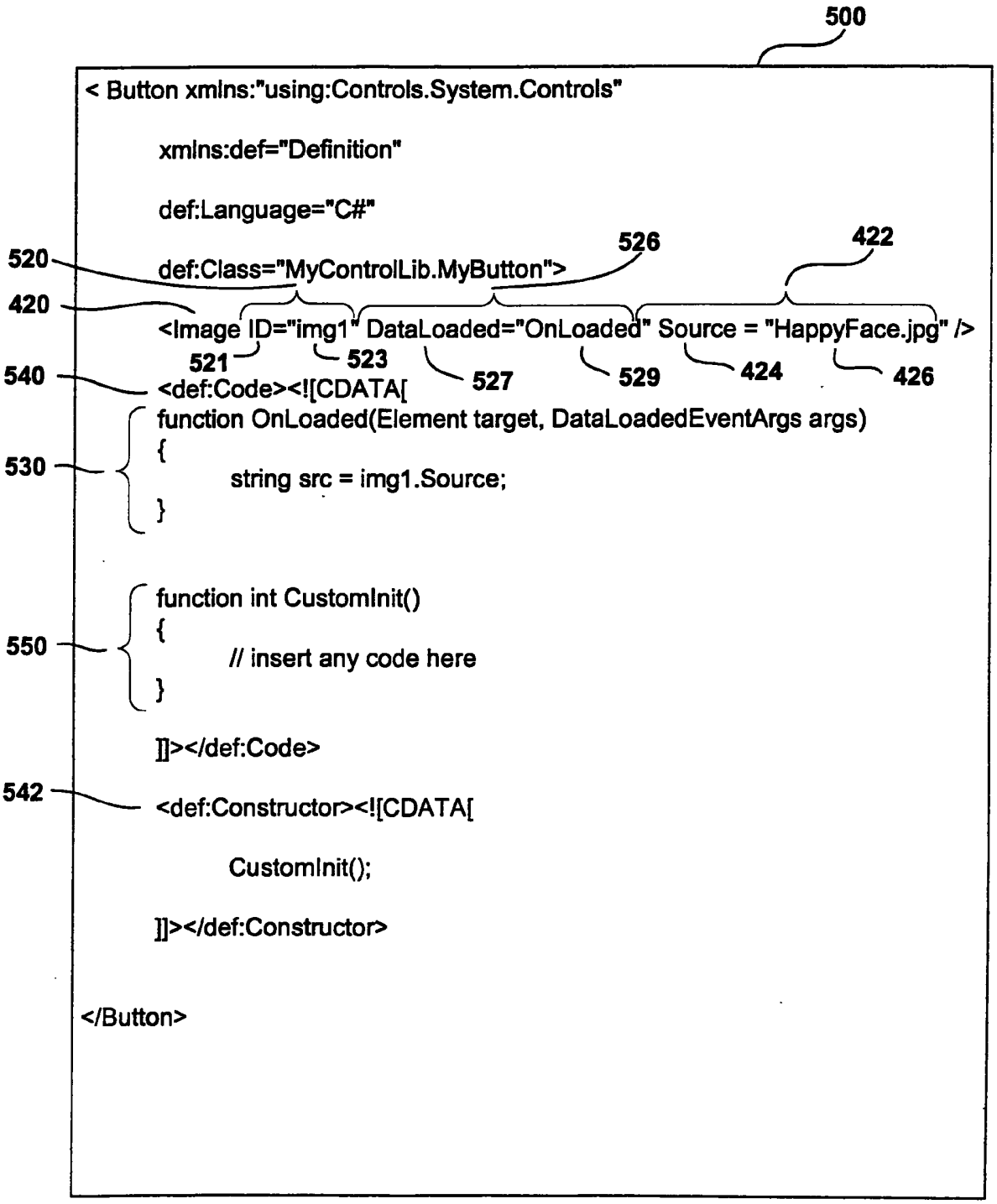

500
< Button xmlns:"using:Controls.System.Controls"
xmlns:def="Definition"
def:Language="C#"
520
def:Class="MyControlLib.MyButton">
526
422
420
<Image ID="img1" DataLoaded="OnLoaded" Source = "HappyFace.jpg" />
521
523
527
529
424
426
540
<def:Code><![CDATA[
function OnLoaded(Element target, DataLoadedEventArgs args)
{
530
string src = img1.Source;
}
function int CustomInit()
{
550
// insert any code here
}
]]></def:Code>
542
<def:Constructor><![CDATA[
CustomInit();
]]></def:Constructor>
</Button>

Fig. 5

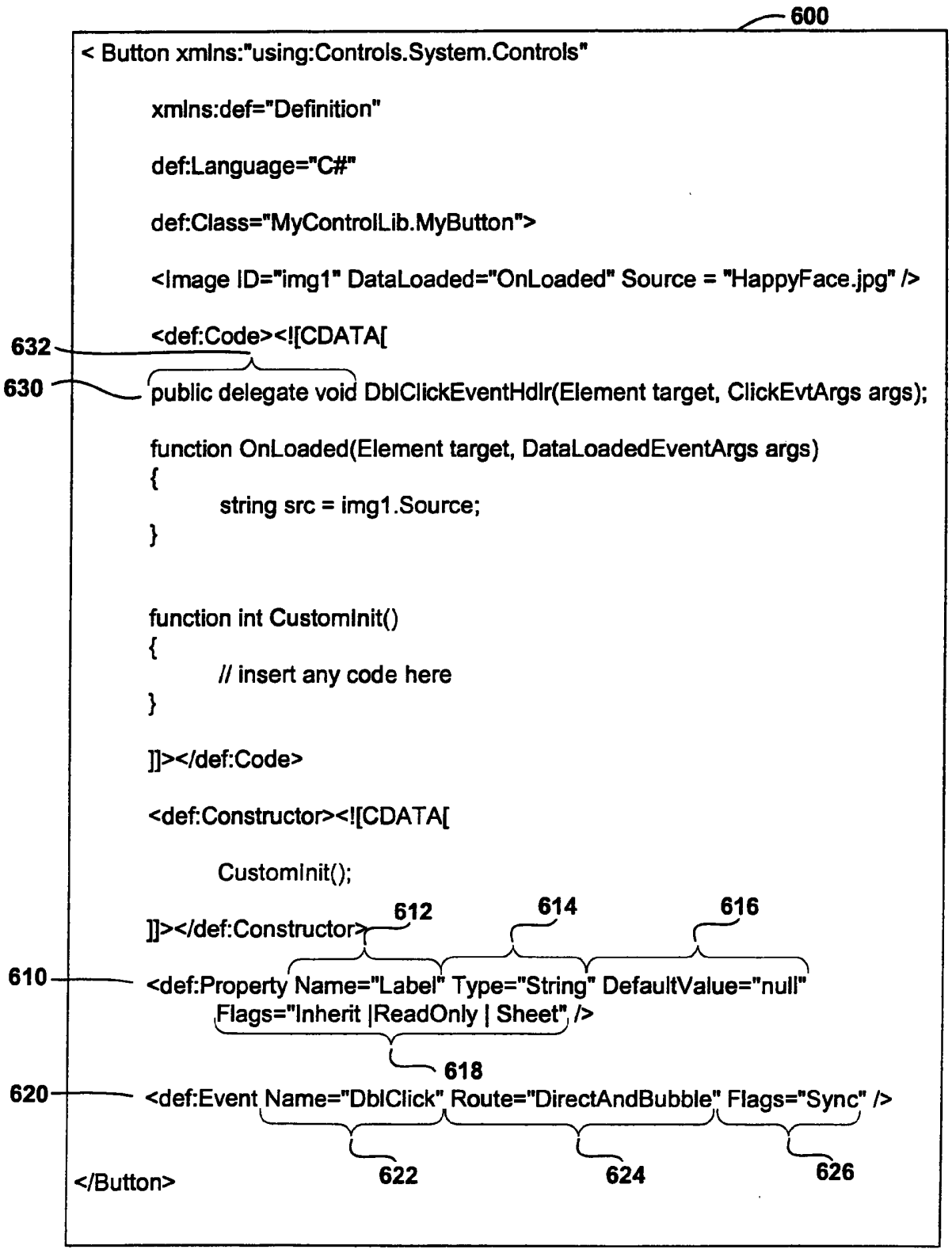

600
< Button xmlns:"using:Controls.System.Controls"
xmlns:def="Definition"
def:Language="C#"
def:Class="MyControlLib.MyButton">
<Image ID="img1" DataLoaded="OnLoaded" Source = "HappyFace.jpg" />
<def:Code><![CDATA[
632
630
public delegate void DblClickEventHdlr(Element target, ClickEvtArgs args);
function OnLoaded(Element target, DataLoadedEventArgs args)
{
string src = img1.Source;
}
function int CustomInit()
{
// insert any code here
}
]]></def:Code>
<def:Constructor><![CDATA[
CustomInit();
]]></def:Constructor>
612
614
616
610
<def:Property Name="Label" Type="String" DefaultValue="null"
Flags="Inherit |ReadOnly | Sheet" />
618
620
<def:Event Name="DblClick" Route="DirectAndBubble" Flags="Sync" />
622
624
626
</Button>

Fig. 6

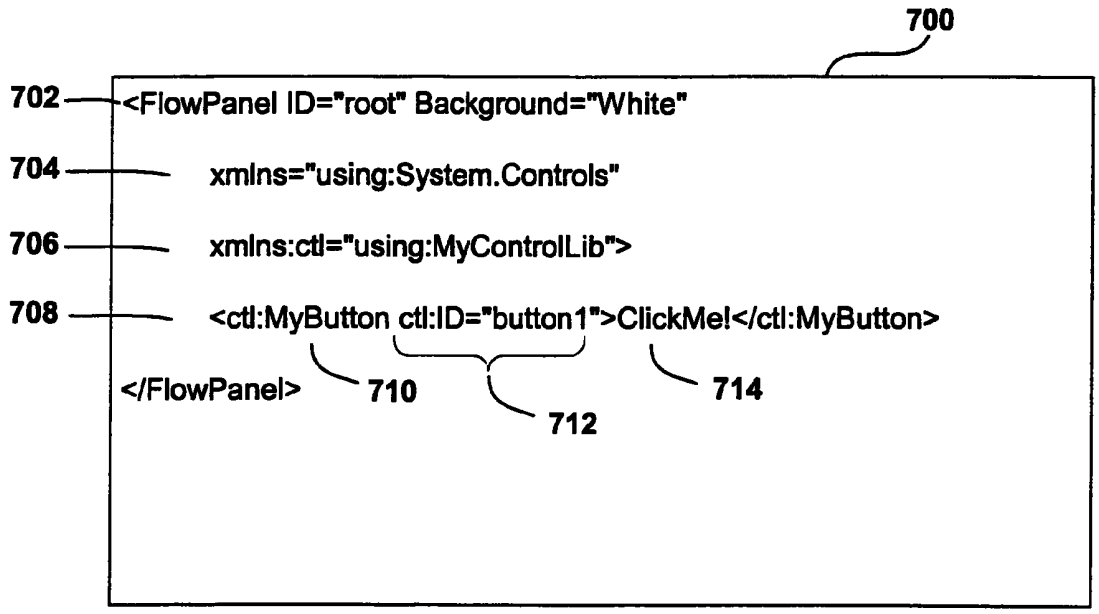

700
702
<FlowPanel ID="root" Background="White"
704
xmlns="using:System.Controls"
706
xmlns:ctl="using:MyControlLib">
708
<ctl:MyButton ctl:ID="button1">ClickMe!</ctl:MyButton>
</FlowPanel>
710
712
714

Fig. 7

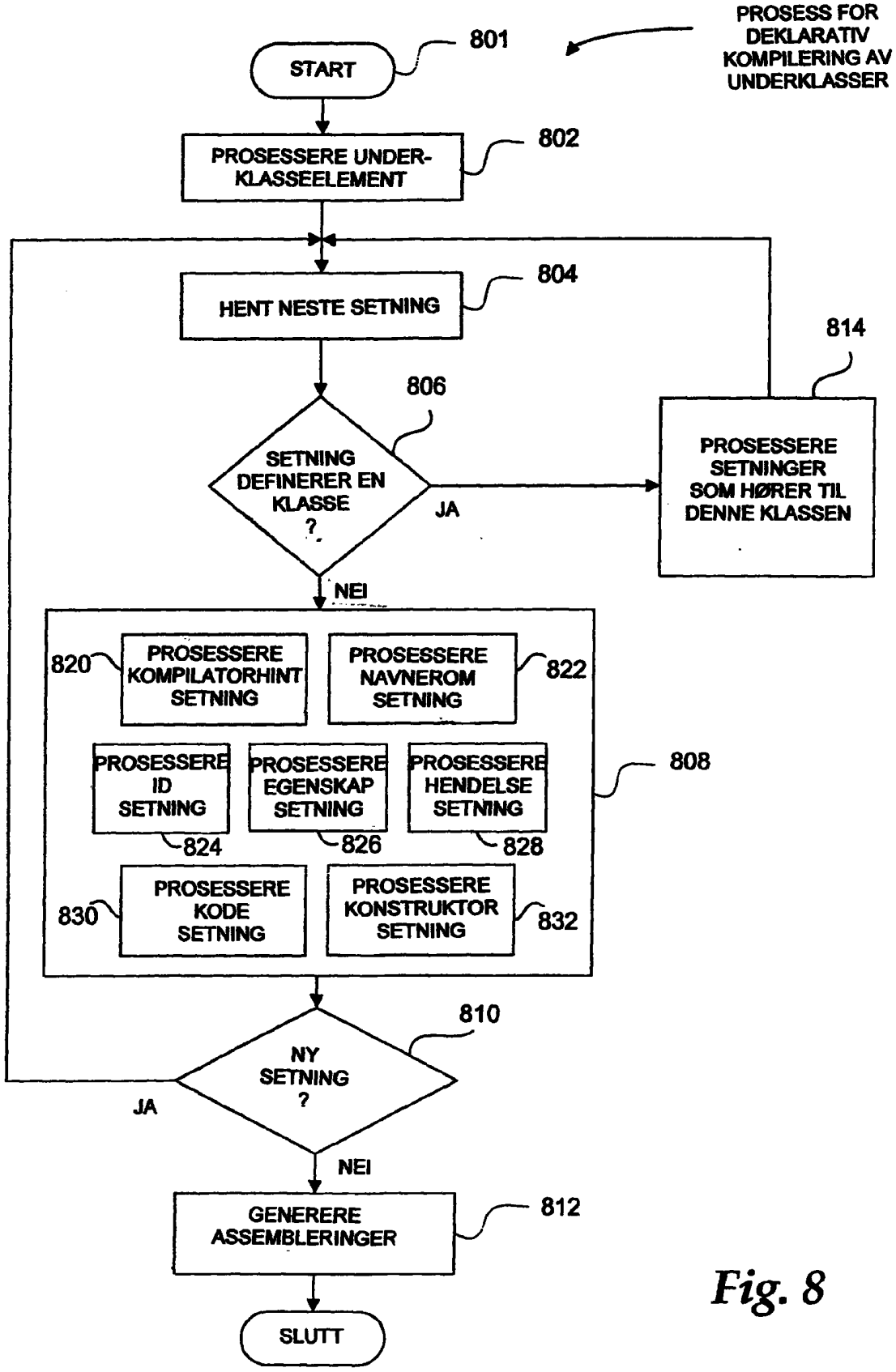

PROSESS FOR DEKLARATIV KOMPILERING AV UNDERKLASSER
START
801
PROSESSERE UNDER-KLASSEELEMENT
802
HENT NESTE SETNING
804
SETNING DEFINERER EN KLASSE ?
806
JA
PROSESSERE SETNINGER SOM HØRER TIL DENNE KLASSEN
814
NEI
PROSESSERE KOMPILATORHINT SETNING
820
PROSESSERE NAVNEROM SETNING
822
PROSESSERE ID SETNING
824
PROSESSERE EGENSKAP SETNING
826
PROSESSERE HENDELSE SETNING
828
808
PROSESSERE KODE SETNING
830
PROSESSERE KONSTRUKTOR SETNING
832
NY SETNING ?
810
JA
NEI
GENERERE ASSEMBLERINGER
812
SLUTT

Fig. 8

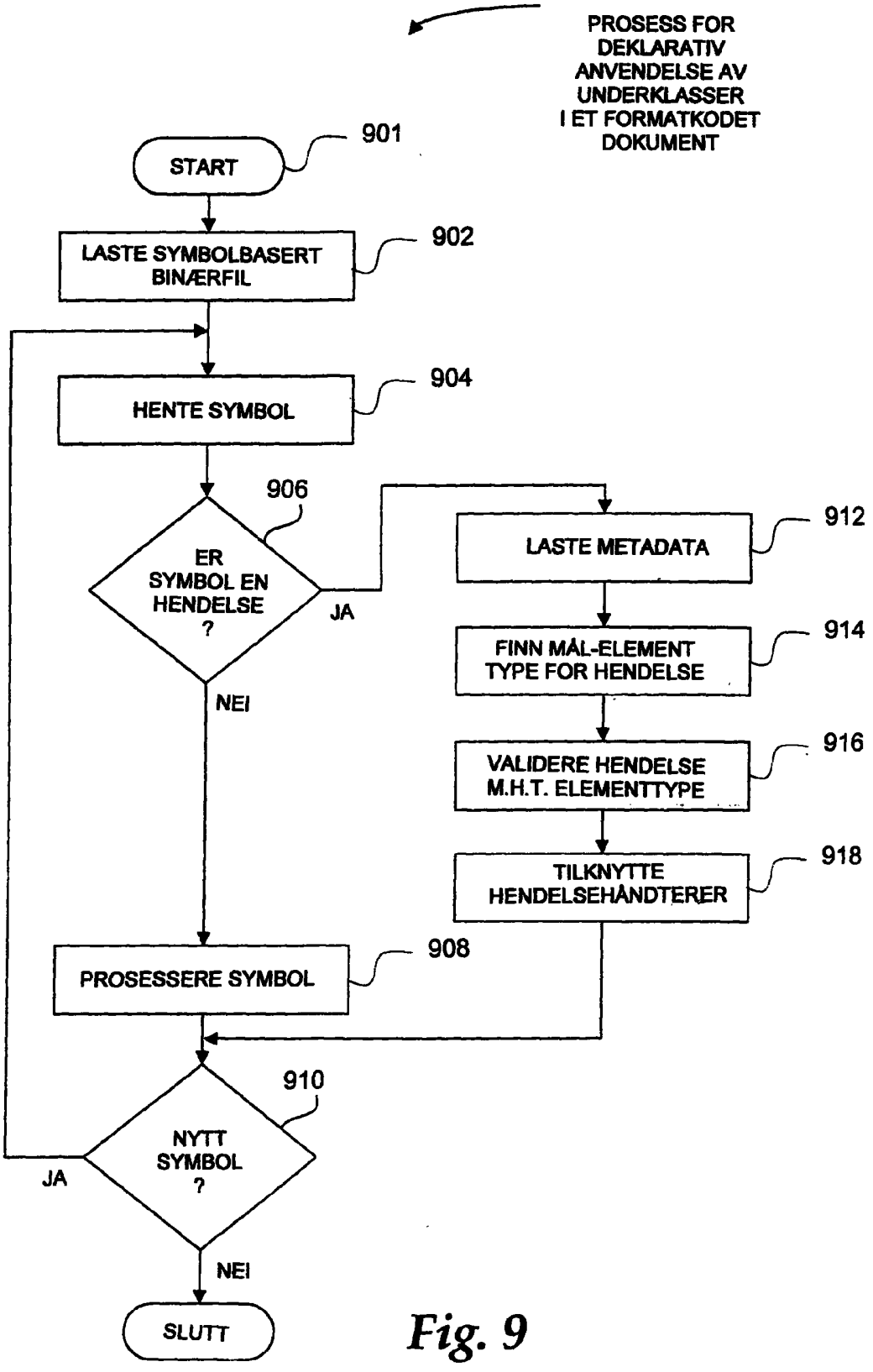
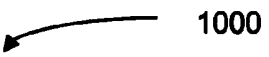

PROSESS FOR DEKLARATIV ANVENDELSE AV UNDERKLASSER I ET FORMATKODET DOKUMENT
START
901
LASTE SYMBOLBASERT BINÆRFIL
902
HENTE SYMBOL
904
ER SYMBOL EN HENDELSE ?
906
JA
NEI
LASTE METADATA
912
FINN MÅL-ELEMENT TYPE FOR HENDELSE
914
VALIDERE HENDELSE M.H.T. ELEMENTTYPE
916
TILKNYTTE HENDELSEHÅNDTERER
918
PROSESSERE SYMBOL
908
NYTT SYMBOL ?
910
JA
NEI
SLUTT

Fig. 9




```
1  namespace MyControlLib {
2    using System.Controls;
3    using System;
4
5    public class MyButton : System.Controls.Button {
6
7    private System.Controls.Image img1;
8
9    public static System.DynamicProperty LabelProperty =
10     System.ComponentModel.Properties.PropertyManager.RegisterProperty(
11       "Label", ((PropertyFlags.Inherit | PropertyFlags.ReadOnly) | PropertyFlags.Sheet),
12       typeof(String), null, typeof(MyButton), null, typeof(MyButton), 0);
13
14   public static System.DynamicEvent DblClickEvent =
15     System.EventManager.RegisterEvent("DblClick", EventRoute.DirectAndBubble,
16     EventFlags.Sync, typeof(DblClickEventHandler), typeof(MyButton));
17
18     function OnLoaded(Element target, DataLoadedEventArgs args)
19     {  string src = img1.Source;              }
20
21     function int CustomInit()                 { }
22
23     public MyButton(System.Windows.Element parent) : base(parent)
24     {  this.__Initialize_This();              }
25
26     public MyButton() :  this(null)           { }
27
28     public String Label {
29        get { return ((String)(this.GetValue(MyButton.LabelProperty)));       }
30     }
31
32     private void __Initialize_This() {
33        CustomInit();
34        System.Controls.Button _Button_1_ = this;
35        System.Controls.Image _Image_2_ = new System.Controls.Image();
36        this.img1 = _Image_2_;
37        _Image_2_.SetValue(System.Element.IDProperty, "img1");
38        _Image_2_.AddHandler(System.Controls.Image.DataLoadedEvent, new
39                    System.Controls.DataLoadedEventHandler(this.OnLoaded),
40                    EventStage.Bubble, EventHandled.Unhandled, true);
41        _Image_2_.SetValue(System.Controls.Image.SourceProperty, "HappyFace.jpg");
42        _Button_1_.Elements.InsertBefore(null, _Image_2_);
43        }
44 } }
```

Fig. 10