

Laboration 1

- Utvecklingsprocessen av ett tolkat diagram-språk och användargränssnitt

Björn Granholm



<2025:05>

Handledare: Joakim Isaksson

INNEHÅLLSFÖRTECKNING

INNEHÅLLSFÖRTECKNING	2
1. INLEDNING	3
1.1 Bakgrund	3
1.2 Syfte	3
1.3 Omfattning	3
2. SPRÅK DESIGN	4
2.1 Grammatik och syntax	4
2.2 Utbyggnadsmöjligheter	5
3. LEXER DESIGN	5
3.1 Token logik	5
3.2 Unicode	6
4. PARSER DESIGN	7
4.1 Översikt av parsers arkitektur	7
4.2 Parser logik	8
4.3 Hantering av attribut och standardvärden	8
4.4 Felrapportering	9
5. RENDERING AV DIAGRAM	9
5.1 Renderer-arkitektur	9
5.2 Stöd för olika layout-algoritmer	10
5.3 Färg-, form- och text-attribut	10
5.4 Svg generering	11
6. ANVÄNDARGRÄNSSNITT	12
6.1 Kommandotolk	12
6.2 Terminalbaserat textgränssnitt	13
7. SAMMANFATTNING	14
KÄLLOR	14

1. INLEDNING

1.1 Bakgrund

I takt med att informationsmängder växer blir det allt viktigare att kunna visualisera komplexa samband på ett överskådligt sätt. Ett domänspecifikt språk (DSL)(Wikipedia contributors, 2025b) för beskrivning av diagram gör det möjligt för användare att skriva textbaserade beskrivningar som automatiskt omvandlas till grafiska representationer. Genom att ta fram en effektiv kedja av lexikalisk och syntaktisk analys, tolkning och rendering kan vi skapa ett flexibelt verktyg för såväl tekniska experter som icke-specialister.

1.2 Syfte

Syftet med denna rapport är att presentera designen och implementationen av en komplett process som omvandlar text till färdiga diagram. Fokus ligger på att beskriva hur varje delkomponent samarbetar för att uppnå hög användarvänlighet, lätt underhållbarhet och god prestanda.

1.3 Omfattning

Rapporten täcker följande delar:

- Definition av det domänspecifika språket inklusive grammatik och tokens
- Lexikalisk analys (lexer)(Improve, 2015a) och syntaktisk analys (parser)(Improve, 2015b)
- Tolk (interpreter) som bygger upp och utvärderar ett abstrakt syntaxträd (AST)(Wikipedia contributors, 2025a)
- Renderer för diagrammet med stöd för SVG
- Användargränssnitt i form av kommandotolk (CLI) och terminal gränssnitt(TUI) för interaktion med verktyget

Samtliga delar är implementerade i Go(*Documentation*, n.d.) och designade för att vara modulära och utbyggbara.

2. SPRÅK DESIGN

I detta avsnitt beskrivs de grundläggande principerna för det domänspecifika språket (DSL) som används för att definiera diagram i textform. Fokus ligger på grammatisk struktur, nyckelord, tokens och möjligheter till framtida utvidgning.

2.1 Grammatik och syntax

Språket bygger på en kontextfri grammatik (CFG)(dixitaditya2001 Follow Improve, 2023) och är linjärt, enkelt och läsbart. Den grundläggande strukturen för ett diagram ser ut så här:

```
Shell
diagram <diagramtyp> (valfria attribut) {
    <statement>*
}
```

Figur 1. Strukturen för en diagramdefinition

- diagram är ett nyckelord som inleder definitionen
- <diagramtyp> anger vilket slags diagram(flowchart eller tree)
- (valfria attribut) omfattar globala attribut som layout=vertical
- {.. } omsluter diagraminnehållet
- <statement> är antingen en nod- eller kantdeklaration

Exempel:

```
Shell
node A "Start"
```

Figur 2. Nod med ID A och etiketten "Start"

```
Shell
```

```
A -> B "Next"
```

Figur 3. Kant från nod A till nod B med etiketten "Next"

2.2 Utbyggnadsmöjligheter

DSL:en är designad med framtida expansion i åtanke. Den är modulär och kan utökas utan att bryta befintlig syntax. Exempel på möjliga utbyggnader:

- **Nya nyckelord** som cluster eller note kan introduceras genom att uppdatera lexer/token-listan.
- **Utökad grammatik** kan lägga till stöd för till exempel grupperade noder, flera diagram i samma fil, eller interaktiva element.
- **Avancerade attribut** som stil, metadata eller layoutlogik kan införas successivt.

3. LEXER DESIGN

I detta avsnitt beskrivs lexerns arkitektur och hur den omvandlar text till en ström av tokens, med fokus på token logik, Unicode-stöd och felhantering

3.1 Token logik

Lexern arbetar med en loop som itererar genom indatasträngen tecken för tecken. Den använder Unicode-medvetna funktioner för att identifiera bokstäver, siffror och blanksteg. Baserat på det aktuella tecknet bestämmer lexern hur tecknen ska grupperas till ett token.

Viktiga aspekter av tokeniseringsprocessen:

- Blanksteg ignoreras.
- Ord identifieras som antingen nyckelord eller identifierare, beroende på om de matchar fördefinierade ord.

- Strängar fångas mellan citattecken.
- Fleratecken-symboler som -> upptäcks med hjälp av framåtblick (lookahead).
En-teckens-symboler hanteras uttryckligen.
- Varje gång ett token känns igen, läggs det till i en tokenlista som returneras till parseern när hela indata har bearbetats.

Tabell 1. Token typer

Token typ	Beskrivning
TOKEN_KEYWORD	Reserverat för ord som diagram och node
TOKEN_IDENTIFIER	Diagramstyper, nod-id eller attributvärden
TOKEN_STRING	Text inom citattecken
TOKEN_ARROW	Pilarna emellan Nod id till Nod id
TOKEN_LBRACE	Vänster måsvinge { som påbörjar blocket
TOKEN_RBRACE	Höger måsvinge } som avslutar blocket
TOKEN_SYMBOL	Tecken som =, (,) och ,
TOKEN_EOF	Markerar slutet på filen

3.2 Unicode

För att korrekt hantera multibyte-tecken konverteras beskrivningen till en slice av runor:

```
Go
runes := []rune(input)
length := len(runes)
```

Figur 4. Konvertering av sträng till en slice

Alla positioner och lookahead tar hänsyn till index. Vid klassificering används Go's unicode-paket(*Unicode*, n.d.):

```
Go
if unicode.IsLetter(c) { ... }
if unicode.IsDigit(c)  { ... }
```

Figur 5. Exempel på tecken klassificering med unicode-stöd

4. PARSER DESIGN

Parseern tar en lista av tokens från lexern och omvandlar dem till en intern datastruktur (AST) som representerar ett diagram med noder och kanter. Parseern är sekventiellt uppbyggd med en central funktion och logik som hanterar varje delstruktur direkt i samma metod. Den implementeras som en enkel, funktionell *recursive descent*-parser utan separata delparser för varje grammatiskt element.

4.1 Översikt av parserns arkitektur

Parseern utgår från funktionen `Parse()` som initierar ett parser-objekt och anropar `parseDiagram()`. Parsingprocessen sker genom att sekventiellt konsumera tokens och fatta beslut baserat på deras typ och innehåll. Funktioner som `currentToken()`, `advance()` och `match()` används för att navigera och kontrollera indata.

Diagrammet representeras som ett objekt av typen `diagram`, som innehåller namn, layoutinställning, samt listor av node och edge. Parseern kontrollerar stegvis att tokenföljden följer den förväntade strukturen: ett diagram-nyckelord, följt av typnamn, eventuella attribut inom parentes och ett block `{ }` som innehåller noder och kanter.

4.2 Parser logik

Parseern implementerar parsing logiken direkt i `parseDiagram()` utan att bryta ut separata metoder för noder eller kanter. Exempel på strukturell parsing:

- Om ett node-nyckelord hittas:
 - Nästa två token tolkas som nodens ID och etikett.
 - Attribut inuti parenteser parsas inline.
 - Standardvärden används om attribut saknas (t.ex. färg och form).
- Om ett identifierar-token följs av `->`, behandlas det som en kant:
 - Från- och till-nod samt eventuell etikett parsas.
 - Även här kan kantens attribut specificeras inom parenteser.

Kodflödet hanterar varje element direkt, vilket ger enkel och tydlig logik, men begränsad modularitet.

4.3 Hantering av attribut och standardvärden

Attribut skrivs som nyckel-värde-par inuti parenteser. Varje attribut mappas till ett fält i motsvarande AST-nod. Parseern är flexibel, om inga attribut anges används definierade standardvärden i renderingsfasen.

Exempel på parserns logik:

- Om noden node A "Start" saknar attribut tilldelas t.ex. standardfärg och form.
- Om en kant saknar färg eller bredd antas exempelvis `color=black` och `width=1`.

Denna strategi gör att användaren kan skriva minimalt men ändå få fullt fungerande output.

4.4 Felrapportering

Parsern innehåller grundläggande felhantering genom `fmt.Errorf(...)` (*Fmt Package - Fmt - Go Packages*, n.d.). Vid avvikelser från den förväntade tokenstrukturen genereras ett felmeddelande med en beskrivning, t.ex.:

- Saknat nyckelord diagram
- Ogiltig diagramtyp (kontrolleras mot en lista av tillåtna typer)
- Saknad likhetstecken i attribut
- Felplacerade måsvingar `{}, }` eller parenteser `(,)`

Parsern avbryter tolkningen vid första allvarliga fel och är inte byggd för återhämtning eller fortsatt tolkning efter fel. Det finns därmed potential för framtida förbättringar inom mer robust felsökning eller användarfeedback.

5. RENDERING AV DIAGRAM

När ett diagram har parserats till ett AST går informationen vidare till renderings delen. Renderaren ansvarar för att omvandla nod- och kantdata till en SVG-bild. Denna sektion beskriver renderers arkitektur, layout system, grafiska attribut och generering av SVG.

5.1 Renderer-arkitektur

Renderern är fristående från parsern och DSL:en, och bygger helt på datastrukturen `Diagram` som innehåller noder, kanter och attribut. Den är uppdelad i tre logiska steg:

- **Initiering:** Skapar en SVG-yta (canvas) med rätt dimensioner baserat på layout och antal element.
- **Layout:** Bestämmer koordinater för varje nod med hjälp av layoutfunktioner:
 - `ComputeLayout()` – horisontell layout (default)
 - `ComputeVerticalLayout()` – vertikal layout
 - `ComputeTreeLayout()` – trädstruktur via BFS

- **Ritning:** Genererar SVG-element för noder (rektanglar eller ellipser), textetiketter och linjer med pilar för kanter. Varje element får färg, form och stil baserat på attribut eller standardvärden.

För närvarande stöds endast **SVG-export** via funktionen `RenderSVG()`. Den genererade SVG-strängen skrivs till fil med .svg-ändelse i en separat output-mapp

5.2 Stöd för olika layout-algoritmer

Diagrammet kan visas i olika strukturella layouter beroende på attribut som `vertical`, `horizontal`. `Horizontal` är default värde på diagrammen.

De viktigaste algoritmerna är:

- **Vertikal layout:** Placerar noder i en kolumn, används ofta för flödesdiagram.
- **Horisontell layout:** Noder läggs i en rad, t.ex. tidslinjer.

Layouten beräknar nodernas positioner innan de ritas, vilket säkerställer att diagrammet blir balanserat och läsbart oavsett antal objekt.

5.3 Färg-, form- och text-attribut

Varje nod och kant kan ges grafiska egenskaper via attribut:

- **Färg:** Både bakgrund (`color`) och text (`text`) kan sättas, t.ex. `color=lightgreen`.
- **Form:** Noder kan visas som `rectangle` och `ellipse`
Text: Etiketter sätts med strängar (t.ex. "Start"), placeras centrerat i noder eller längs kanter.
- **Kantstil:** Bredd (`width`), färg och typ (`streckad`, `pilspets`) anges direkt i DSL:en.

5.4 Svg generering

Diagrammen exporteras som SVG (Scalable Vector Graphics)(*Scalable Vector Graphics (SVG)* 2, n.d.), vilket är ett XML-baserat vektorformat som lämpar sig väl för webbläsare och dokument. Ingen mellanliggande grafikmotor används, istället byggs SVG-koden upp direkt i programmet som en sträng.

Renderern konstruerar SVG-element för varje del av diagrammet:

- **Noder** genereras som `<rect>`, `<ellipse>` eller andra geometriska taggar beroende på nodens `shape`-attribut. Position, storlek, färg och fyllning styrs via attribut som `x`, `y`, `width`, `height`, `fill`, `stroke`.
- **Textetiketter** placeras med `<text>`-element med justerade koordinater och färg via `fill`.
- **Kanter** ritas med `<line>` eller `<path>` beroende på stil, och etiketter för stegnamn ritas separat ovanpå linjerna.

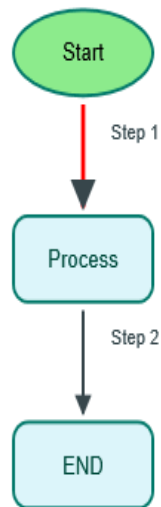
Exempel på en nod i svg:

XML

```
<ellipse cx="100" cy="50" rx="40" ry="20" fill="lightgreen" stroke="black" />
<text x="100" y="55" text-anchor="middle" fill="black">Start</text>
```

Figur 6. En nod representerad som en ellips med etiketten "Start", centrerad vid koordinaterna (100, 50), med ljusgrön fyllning och svart kantlinje. Texten är centrerad både horisontellt och vertikalt inom noden.

Den färdiga SVG-strängen ramas in av ett `<svg>`-taggpar med definierad `width`, `height` och `xmlns`. Filen skrivs sedan ut direkt till filsystemet som `.svg`.



Figur 7. Exempel på ett färdigt diagram

6. ANVÄNDARGRÄNSSNITT

Diagramverktyget erbjuder två användarlägen:

- Kommandotolk (CLI)
- Terminalbaserat textgränssnitt (TUI).

Båda lägena är minimalistiska och fokuserar på enkel inmatning och generering av .svg-filer från .diag-källfiler.

6.1 Kommandotolk

CLI aktiveras automatiskt om programmet körs med argument. Funktionen RunCLI tar hand om dessa, och tolkar kommandon som:

- **render <fil>**: Tolkar och genererar SVG från en specifik .diag-fil.

- **render-all:** Läser alla .diag-filer i example/ och genererar motsvarande SVG i output/-katalogen.
- Filnamn kontrolleras för korrekt ändelse och existens innan de skickas till renderfunktionen.
- Slutresultatet sparas som output/<filnamn>.svg och tidmätning sker per fil och totalt

6.2 Terminalbaserat textgränssnitt

Om inga kommandoradsargument anges startas det interaktiva terminalgränssnittet, byggt med biblioteket Bubble Tea(*Bubbletea: A Powerful Little TUI Framework* 🍵, n.d.).

Funktionerna RunTUI och InitialModel laddar .diag-filer från example/-katalogen och presenterar en meny i terminalen:

- **Menyläge:** Visar alternativ som "Render from example", "Render all diagrams", "Quit".
- **Filväljarläge:** Användaren kan bläddra mellan tillgängliga .diag-filer med piltangenter eller j/k, och trycka Enter för att rendera vald fil.
- **Statusrad:** Spinner och statusmeddelanden visar pågående aktivitet.

Gränssnittet använder biblioteket lipgloss(*Lipgloss: Style Definitions for Nice Terminal Layouts* 💋, n.d.) för färg och layout, och erbjuder ett stilrent terminal-UI med ramade boxar, färgkodade val och övergångseffekter.

TUI-läget är idealiskt för manuell användning och demonstration, medan CLI lämpar sig för automatisering och större volymer. Båda lägena använder samma parser och renderer under huven.

7. SAMMANFATTNING

Detta projekt har visat hur ett tolkningsbart domänspecifikt språk (DSL) kan utformas och implementeras för att beskriva och visualisera diagram med hög flexibilitet. Genom att kombinera lexikalisk analys, parserlogik, strukturuppbyggnad och grafisk rendering i Go har ett komplett system skapats, från textbaserad input till färdig SVG-grafik.

Arbetet har resulterat i ett modulärt verktyg med tydlig uppdelning mellan språkdefinition, analys, rendering och användargränssnitt. Språket är lättförståeligt, men kraftfullt nog att beskriva både flödesdiagram och trädstrukturer med grafiska attribut som färg, form och layout. Felhantering finns på grundläggande nivå, och systemet kan byggas ut med nya token, layout-algoritmer eller gränssnitt utan att påverka befintlig funktionalitet.

Slutligen ger verktyget användaren två interaktionssätt, via kommandorad eller terminalbaserat textgränssnitt vilket gör det lämpligt både för automatisering och manuell användning. Sammantaget demonstrerar projektet hur man kan gå från språkspecifikation till färdig visuell applikation med tydlig kodstruktur och användarvänligt gränssnitt.

KÄLLOR

bubbletea: A powerful little TUI framework 🏗️. (n.d.). Github. Retrieved May 29, 2025, from

<https://github.com/charmbracelet/bubbletea>

dixitaditya2001 Follow Improve. (2023, July 11). *What is context-free grammar?*

GeeksforGeeks. <https://www.geeksforgeeks.org/what-is-context-free-grammar/>

Documentation. (n.d.). Retrieved October 9, 2024, from <https://go.dev/doc/>

fmt package - fmt - Go Packages. (n.d.). Retrieved May 14, 2025, from <https://pkg.go.dev/fmt>


Improve, K. F. (2015a, July 13). *Introduction of lexical analysis*. GeeksforGeeks.

<https://www.geeksforgeeks.org/introduction-of-lexical-analysis/>

Improve, K. F. (2015b, September 22). *Introduction to syntax analysis in compiler design*.

GeeksforGeeks.

<https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/>

lipgloss: Style definitions for nice terminal layouts . (n.d.). Github. Retrieved May 29,

2025, from <https://github.com/charmbracelet/lipgloss>

Scalable vector graphics (SVG) 2. (n.d.). Retrieved May 29, 2025, from

<https://www.w3.org/TR/SVG2/>

Unicode. (n.d.). Retrieved May 14, 2025, from <https://pkg.go.dev/unicode>

Wikipedia contributors. (2025a, March 14). *Abstract syntax tree*. Wikipedia, The Free

Encyclopedia.

https://en.wikipedia.org/w/index.php?title=Abstract_syntax_tree&oldid=1280483463

Wikipedia contributors. (2025b, April 16). *Domain-specific language*. Wikipedia, The Free

Encyclopedia.

https://en.wikipedia.org/w/index.php?title=Domain-specific_language&oldid=12859423