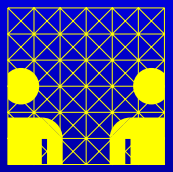
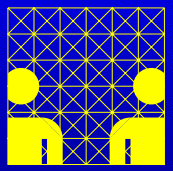


“CLIM” - Der Common LISP Interface Manager

Ein UIMS für Common LISP

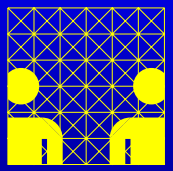
Michael Wessel

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Kognitive Systeme



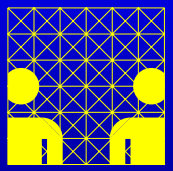
Was ist CLIM?

- Ein UIMS für Common LISP



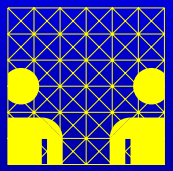
Was ist CLIM?

- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)



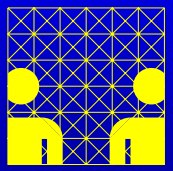
Was ist CLIM?

- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)
  - UIMS $\neq$ Interface Builder



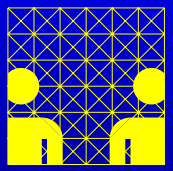
Was ist CLIM?

- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)
  - UIMS $\neq$ Interface Builder
  - Mit Hilfe eines UIMS-Frameworks werden grafische Benutzungsoberflächen auf hoher = problemadäquater Ebene beschrieben



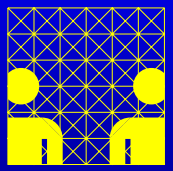
Was ist CLIM?

- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)
  - UIMS $\neq$ Interface Builder
  - Mit Hilfe eines UIMS-Frameworks werden grafische Benutzungsoberflächen auf hoher = problemadäquater Ebene beschrieben
  - $\oplus$ Deklarativität, Abstraktion, Portabilität, Produktivität!



Was ist CLIM?

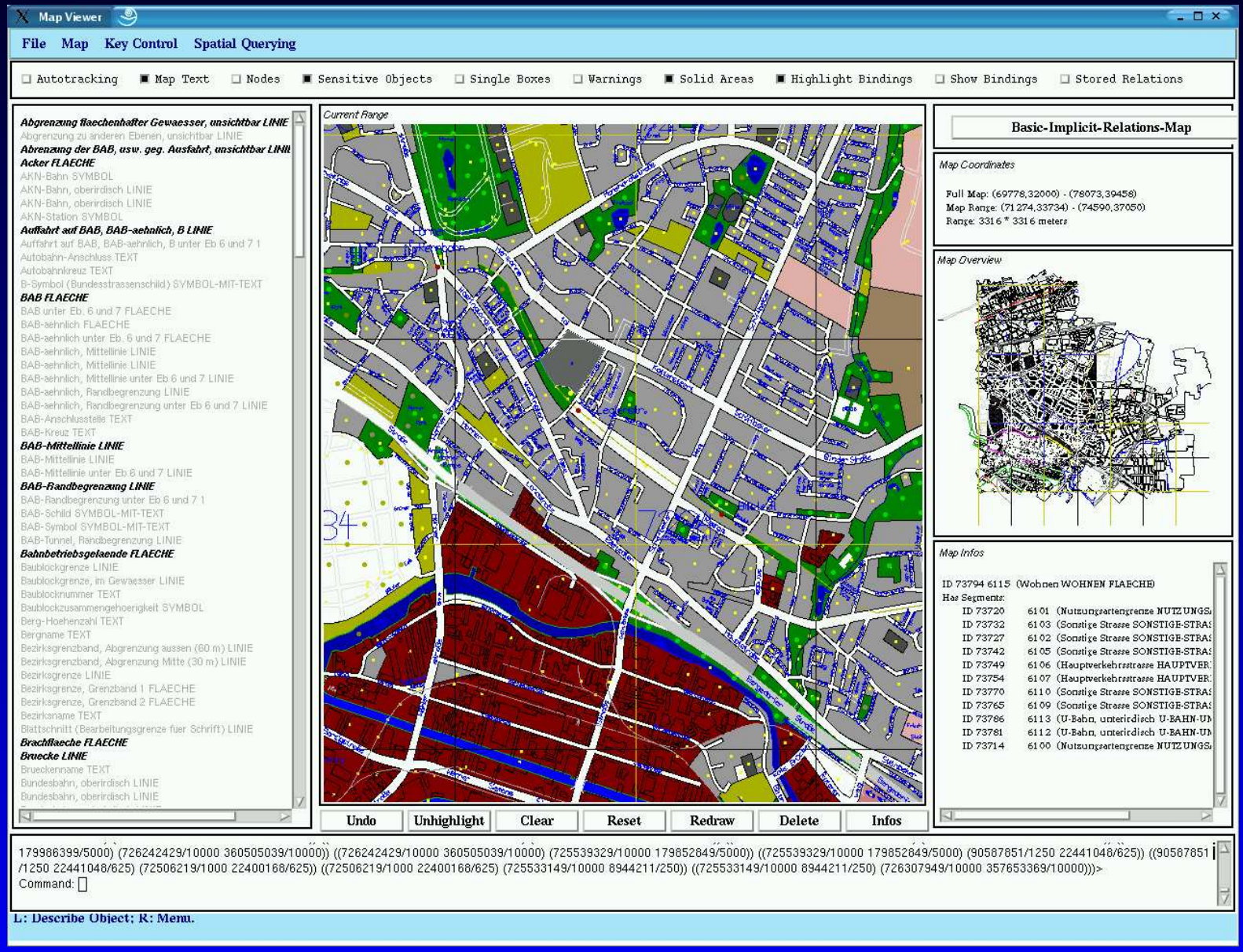
- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)
  - UIMS $\neq$ Interface Builder
  - Mit Hilfe eines UIMS-Frameworks werden grafische Benutzungsoberflächen auf hoher = problemadäquater Ebene beschrieben
  - $\oplus$ Deklarativität, Abstraktion, Portabilität, Produktivität!
  - $\ominus$ Effizienz, Host-System-Spezifizität, Evolution der Zielsysteme

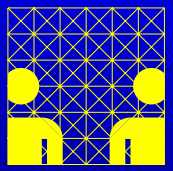


Was ist CLIM?

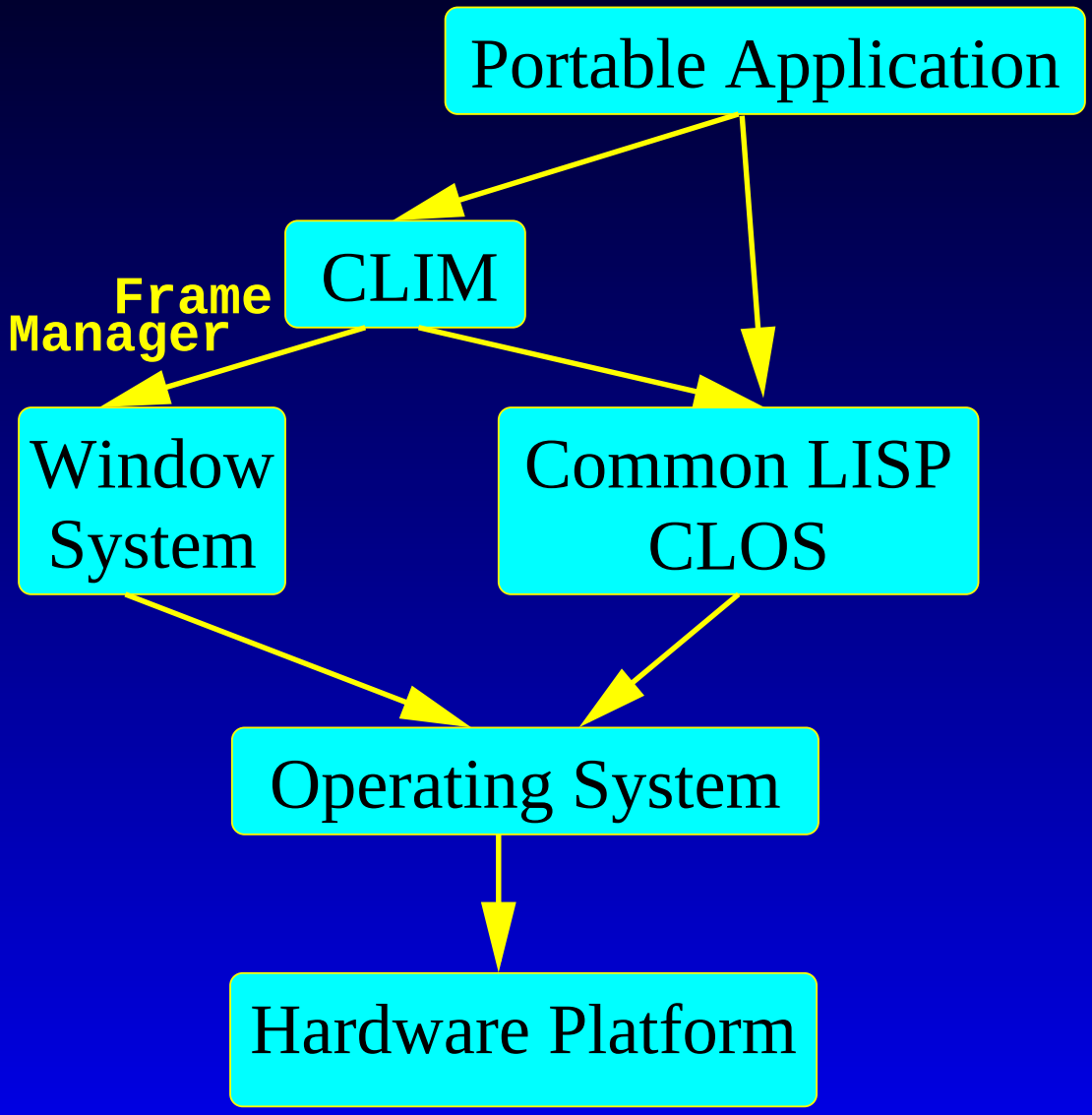
- Ein UIMS für Common LISP
  - UIMS $\neq$ Window System (Windows)
  - UIMS $\neq$ Interface Builder
  - Mit Hilfe eines UIMS-Frameworks werden grafische Benutzungsoberflächen auf hoher = problemadäquater Ebene beschrieben
  - $\oplus$ Deklarativität, Abstraktion, Portabilität, Produktivität!
  - $\ominus$ Effizienz, Host-System-Spezifizität, Evolution der Zielsysteme
  - In CLIM lassen sich ausgefeilte hochfunktionale Benutzungsobeflächen mit minimalstem Aufwand erstellen, aber es fehlt der “neuste Schnick-Schnack”!

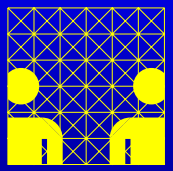
Was ist CLIM?


Map Viewer
File Map Key Control Spatial Querying
Autotracking Map Text Nodes Sensitive Objects Single Boxes Warnings Solid Areas Highlight Bindings Show Bindings Stored Relations
Abgrenzung flaechenhafter Gewaesser, unsichtbar LINIE
Abgrenzung zu anderen Ebenen, unsichtbar LINIE
Abrenzung der BAB, usw. geg. Ausfahrt, unsichtbar LINIE
Acker FLAECHE
AKN-Bahn SYMBOL
AKN-Bahn, oberirdisch LINIE
AKN-Bahn, oberirdisch LINIE
AKN-Station SYMBOL
Auffahrt auf BAB, BAB-aehnlich, B LINIE
Auffahrt auf BAB, BAB-aehnlich, B unter Eb 6 und 7 1
Autobahn-Anschluss TEXT
Autobahnkreuz TEXT
B-Symbol (Bundesstrassenschild) SYMBOL-MIT-TEXT
BAB FLAECHE
BAB unter Eb. 6 und 7 FLAECHE
BAB-aehnlich FLAECHE
BAB-aehnlich unter Eb. 6 und 7 FLAECHE
BAB-aehnlich, Mittellinie LINIE
BAB-aehnlich, Mittellinie LINIE
BAB-aehnlich, Mittellinie unter Eb 6 und 7 LINIE
BAB-aehnlich, Randbegrenzung LINIE
BAB-aehnlich, Randbegrenzung unter Eb 6 und 7 LINIE
BAB-Anschlussstelle TEXT
BAB-Kreuz TEXT
BAB-Mittellinie LINIE
BAB-Mittellinie LINIE
BAB-Mittellinie unter Eb 6 und 7 LINIE
BAB-Randbegrenzung LINIE
BAB-Randbegrenzung unter Eb 6 und 7 1
BAB-Schild SYMBOL-MIT-TEXT
BAB-Symbol SYMBOL-MIT-TEXT
BAB-Tunnel, Randbegrenzung LINIE
Bahnbetriebsgelaende FLAECHE
Baublockgrenze LINIE
Baublockgrenze, im Gewaesser LINIE
Baublocknummer TEXT
Baublockzusammengehoerigkeit SYMBOL
Berg-Hoehenzahl TEXT
Bergname TEXT
Bezirksgrenzband, Abgrenzung aussen (60 m) LINIE
Bezirksgrenzband, Abgrenzung Mitte (30 m) LINIE
Bezirksgrenze LINIE
Bezirksgrenze, Grenzband 1 FLAECHE
Bezirksgrenze, Grenzband 2 FLAECHE
Bezirksname TEXT
Blattschnitt (Bearbeitungsgrenze fuer Schrift) LINIE
Brachflaeche FLAECHE
Bruecke LINIE
Brueckenname TEXT
Bundesbahn, oberirdisch LINIE
Bundesbahn, oberirdisch LINIE
Current Range
Basic-Implicit-Relations-Map
Map Coordinates
Full Map: (69778,32000) - (78073,39458)
Map Range: (71274,33734) - (74590,37050)
Range: 3316 * 3316 meters
Map Overview
Map Infos
ID 73794 6115 (Wohnen WOHNEN FLAECHE)
Has Segments:
ID 73720 6101 (Nutzungsartengrenze NUTZUNGS
ID 73732 6103 (Sonstige Strasse SONSTIGE-STRA
ID 73727 6102 (Sonstige Strasse SONSTIGE-STRA
ID 73742 6105 (Sonstige Strasse SONSTIGE-STRA
ID 73749 6106 (Hauptverkehrsstrasse HAUPTVER
ID 73754 6107 (Hauptverkehrsstrasse HAUPTVER
ID 73770 6110 (Sonstige Strasse SONSTIGE-STRA
ID 73765 6109 (Sonstige Strasse SONSTIGE-STRA
ID 73786 6113 (U-Bahn, unterirdisch U-BAHN-UN
ID 73781 6112 (U-Bahn, unterirdisch U-BAHN-UN
ID 73714 6100 (Nutzungsartengrenze NUTZUNGS
Undo Unhighlight Clear Reset Redraw Delete Infos
179986399/5000) (726242429/10000 360505039/10000)) ((726242429/10000 360505039/10000) (725539329/10000 179852849/5000)) ((725539329/10000 179852849/5000) (90587851/1250 22441048/625)) ((90587851/1250 22441048/625) (72506219/1000 22400168/625)) ((72506219/1000 22400168/625) (725533149/10000 8944211/250)) ((725533149/10000 8944211/250) (726307949/10000 357653369/10000)))>
Command:
L: Describe Object; R: Menu.



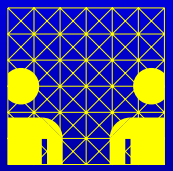
Portabilität durch Abstraktion





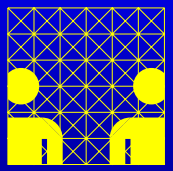
Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung



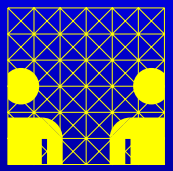
Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für



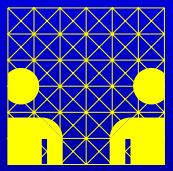
Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)



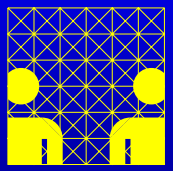
Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)
  - Aufgaben



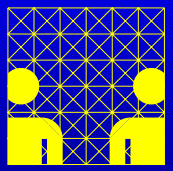
Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)
  - Aufgaben
  - Diskurs



Modellbasierte UIMS

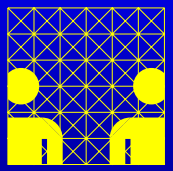
- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)
  - Aufgaben
  - Diskurs
  - evtl. sogar Benutzer, . . .



Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)
  - Aufgaben
  - Diskurs
  - evtl. sogar Benutzer, . . .

⇒ ein modellbasiertes UIMS erzeugt automatisch anhand der Modelle eine grafische Benutzungsschnittstelle für die Anwendung

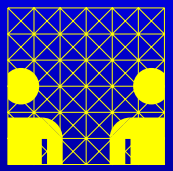


Modellbasierte UIMS

- Zentraler Begriff des Modells der Anwendung
- Problemangepasste deklarativ beschriebene Modelle für
  - Domäne (Objekte der Anwendunge etc.)
  - Aufgaben
  - Diskurs
  - evtl. sogar Benutzer, . . .

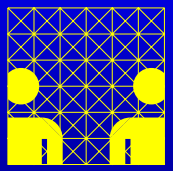
⇒ ein modellbasiertes UIMS erzeugt automatisch anhand der Modelle eine grafische Benutzungsschnittstelle für die Anwendung

⇒ sehr wichtig: multi-modale Abstraktionen für Input (Kommandos) / Output (Präsentationen), Dialoge, . . .



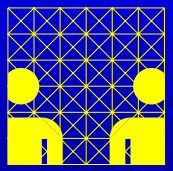
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.



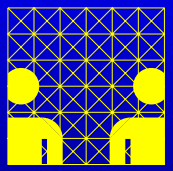
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”



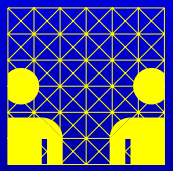
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”
  - Liebermann, ’85: “History of Commands”, getype Präsentationen, Kontextsensitivität von Kommandos, Maus-Sensitivität



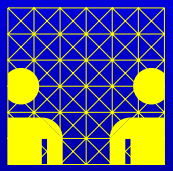
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”
  - Liebermann, ’85: “History of Commands”, getype Präsentationen, Kontextsensitivität von Kommandos, Maus-Sensitivität
- Symbolics Inc., Genera 7.0, 1986: Dynamic Windows



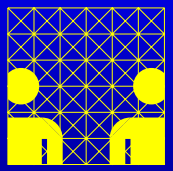
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”
  - Liebermann, ’85: “History of Commands”, getype Präsentationen, Kontextsensitivität von Kommandos, Maus-Sensitivität
- Symbolics Inc., Genera 7.0, 1986: Dynamic Windows
- CLIM 1.0: 1991, Nachfolger bzw. Ergänzung zu Dynamic Windows



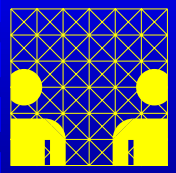
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”
  - Liebermann, ’85: “History of Commands”, getype Präsentationen, Kontextsensitivität von Kommandos, Maus-Sensitivität
- Symbolics Inc., Genera 7.0, 1986: Dynamic Windows
- CLIM 1.0: 1991, Nachfolger bzw. Ergänzung zu Dynamic Windows
- CLIM 2.0: Specification vom 6.5.1992, Scott McKay@Symbolics et al.



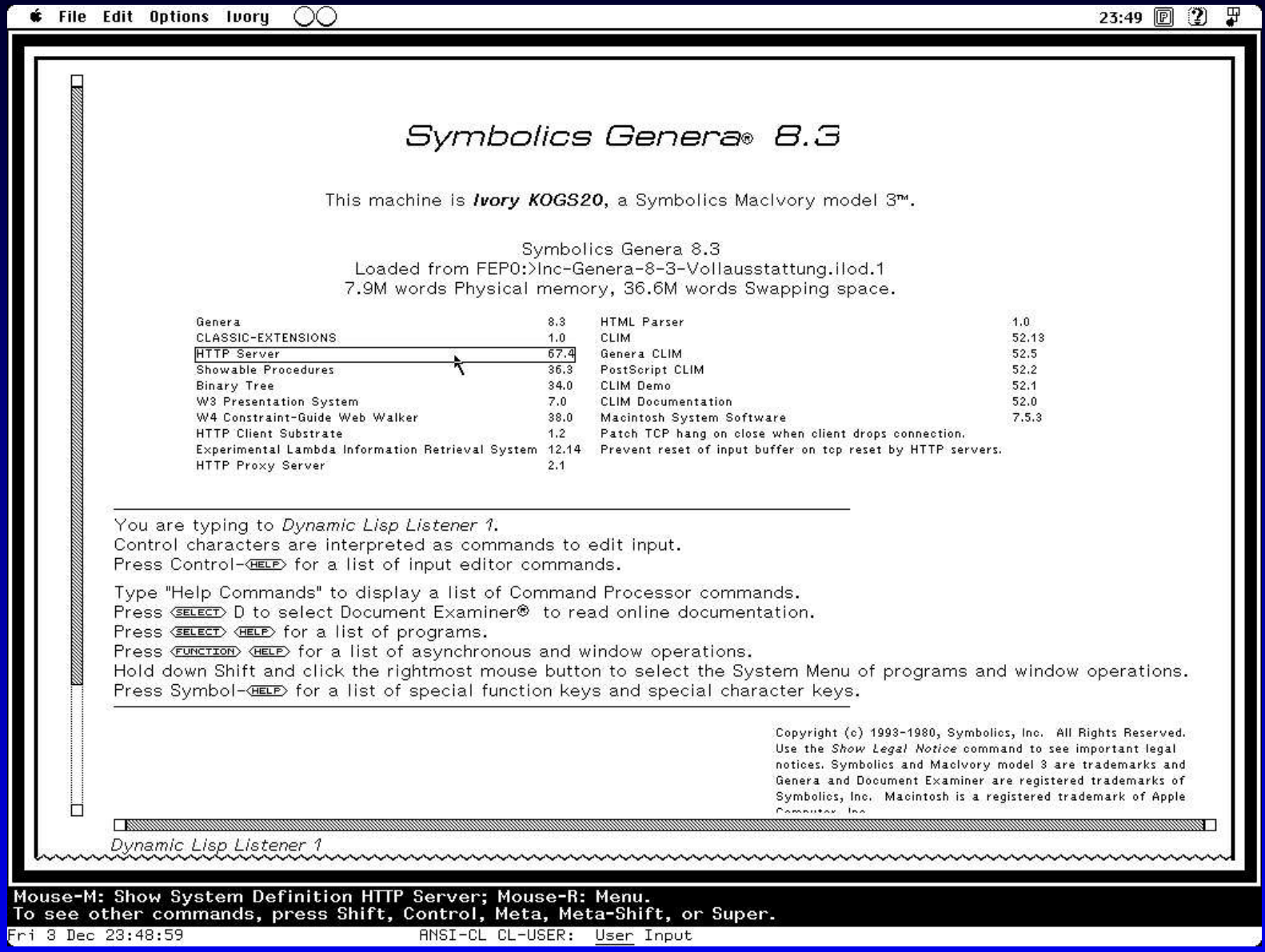
Geschichte von CLIM

- Kern-Konzepte und -Abstraktionen von CLIM wurden bereits zuvor eingeführt, z.B.
  - Ciccarelli, ’84: Prinzip der “Presentations”
  - Liebermann, ’85: “History of Commands”, getype Präsentationen, Kontextsensitivität von Kommandos, Maus-Sensitivität
- Symbolics Inc., Genera 7.0, 1986: Dynamic Windows
- CLIM 1.0: 1991, Nachfolger bzw. Ergänzung zu Dynamic Windows
- CLIM 2.0: Specification vom 6.5.1992, Scott McKay@Symbolics et al.
- Mitte der ’90er: Symbolics ist pleite

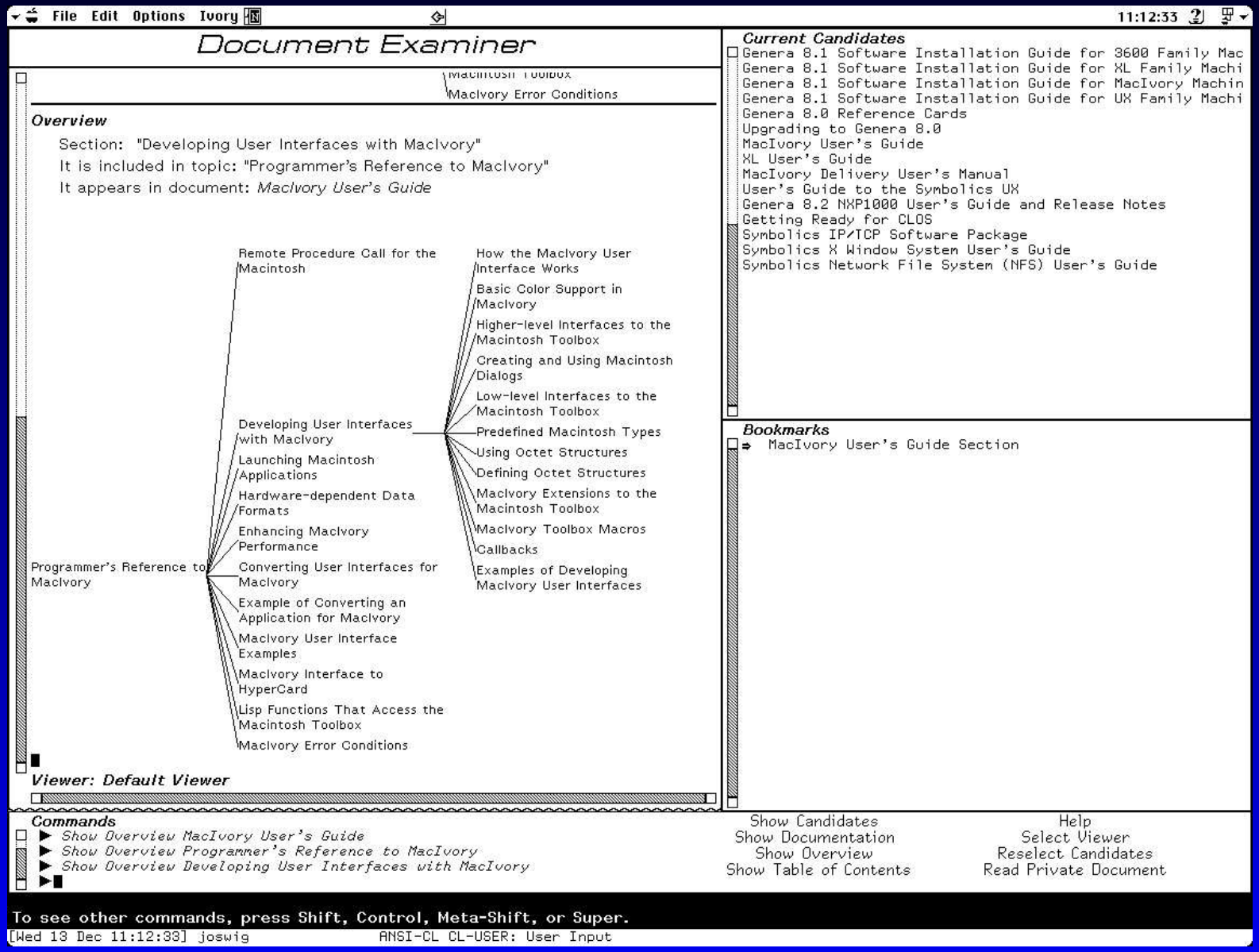


Geschichte von CLIM

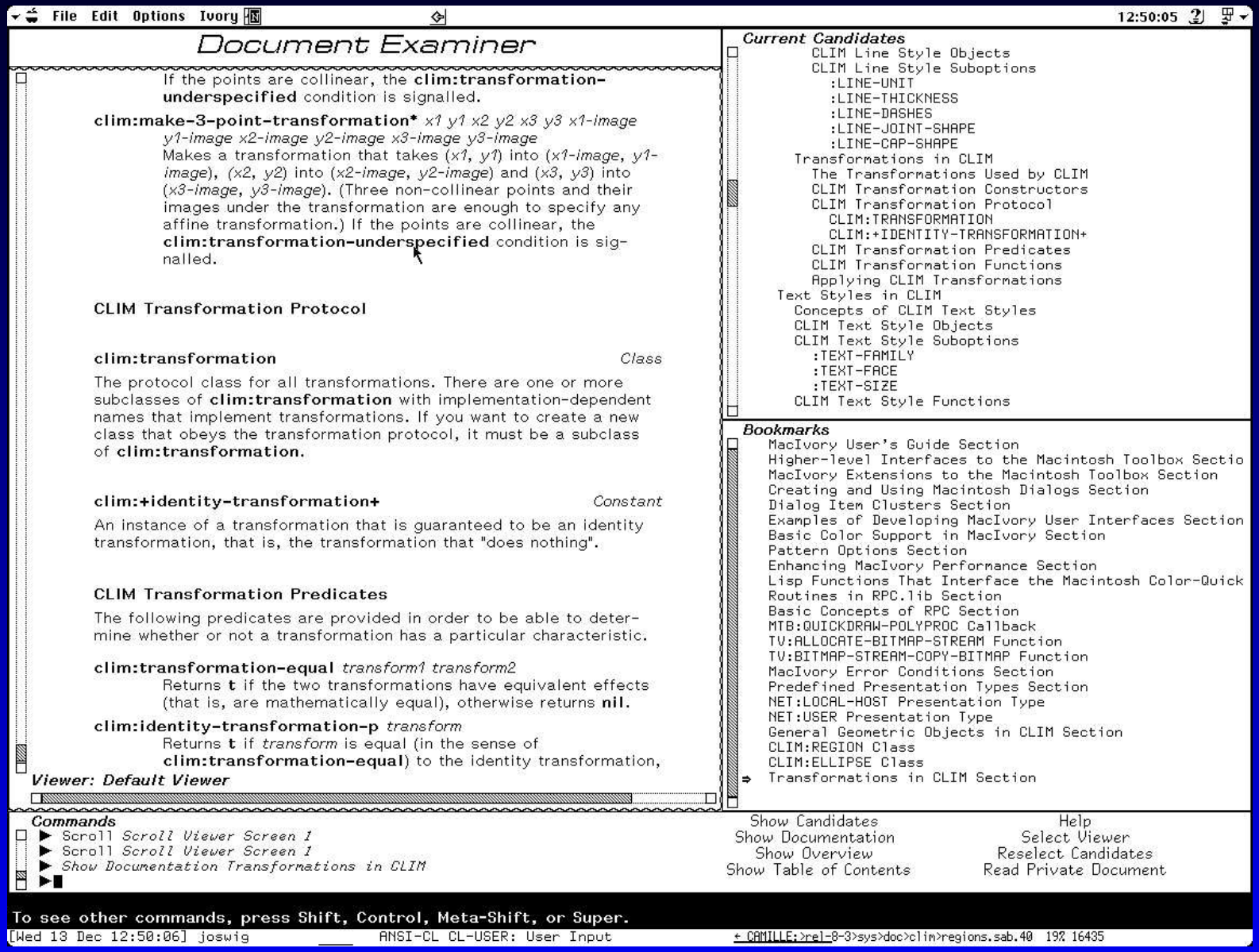
Geschichte von CLIM

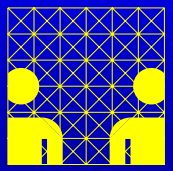


Geschichte von CLIM



Geschichte von CLIM





Geschichte von CLIM

File Edit Options Ivory 20:54

NAOS Event-Recognition

3D-Picture Menu: create animate file

Configuration-Selection Menu: TAF Operations Edit Config Eval Config Config Options

GSB File Menu: load animate save

Working Sheet

Pan and Zoom Sheet

pan zoom total view

Dynamic Lisp Listener

Marked event: (ABBIEGEN DUDU STRASSE-B (17.1 19.0) (18.6 22.0))

```
    TEST:  (PARALLEL DUDU FUSSWEG-1 18.8 21.3)
    TEST:  (PARALLEL DUDU FUSSWEG-1 15.1 17.6)
    TEST:  (PARALLEL DUDU FUSSWEG-2 15.1 18.5)
    TEST:  (PARALLEL DUDU FUSSWEG-3 15.1 18.5)
    TEST:  (PARALLEL DUDU FUSSWEG-5 21.2 25.9)
    ZIEL:  (AUF DUDU FUSSWEG-5 21.2 42.2)
    TEST:  (PARALLEL DUDU STRASSE-H 15.1 18.4)
    TEST:  (PARALLEL DUDU STRASSE-S 21.2 25.9)
    ZIEL:  (AUF DUDU STRASSE-S 21.2 42.2)
    TEST:  (AUF DUDU STRASSE-S 18.5 25.9)
    ZIEL:  (NICHT AUF DUDU STRASSE-S 21.3 25.9)
    TEST:  (ABBIEGEN DUDU STRASSE-B (17.1 19.0) (18.6 22.0))

gefunden:
     (ABBIEGEN DUDU STRASSE-B (17.1 19.0) (18.6 22.0))

 ********************** END OF EVALUATION  **********************
```

NAOS Evaluation Output

Messages

ABBIEGEN	NAHE
ABFAHREN	NEBEN
ANFAHREN	PARALLEL
ANHALTEN	PASSIEREN
AUF	QUER
BEI	RASEN

NAOS Relation Table

NAOS Main-Menu: events focus evaluate unmark inspect protocol FUZZY abort

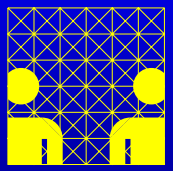
Evaluation-Protocol Switches

Actual Proposition : Yes No
Found Events : Yes No
Evaluation-Sequence : Yes **No**

Clock (s)

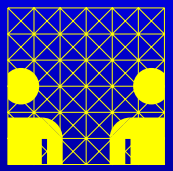
To see other commands, press Shift, Control, Meta-Shift, or Super.

Wed 23 Jun 20:54:19 rm ANSI-CL CL-USER: User Input



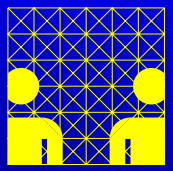
Grundlegende Konzepte

- Anwendungsobjekte



Grundlegende Konzepte

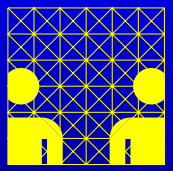
- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...



Grundlegende Konzepte

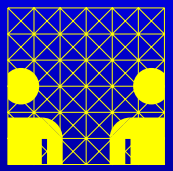
- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...

  $\Rightarrow$ “Modell” in MVC-Terminologie



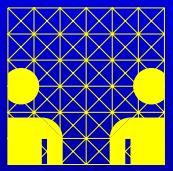
Grundlegende Konzepte

- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...
  - $\Rightarrow$ “Modell” in MVC-Terminologie
  - CLIM: CLOS-Klassen und Methoden (aber nicht zwingend)



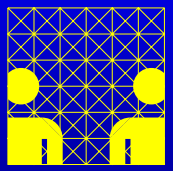
Grundlegende Konzepte

- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...
  - $\Rightarrow$ “Modell” in MVC-Terminologie
  - CLIM: CLOS-Klassen und Methoden (aber nicht zwingend)
- visuelle Repräsentationen der Anwendungsobjekte



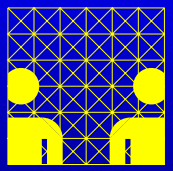
Grundlegende Konzepte

- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...
  - $\Rightarrow$ “Modell” in MVC-Terminologie
  - CLIM: CLOS-Klassen und Methoden (aber nicht zwingend)
- visuelle Repräsentationen der Anwendungsobjekte
  - Display-Objekte $\neq$ Anwendungsobjekte!



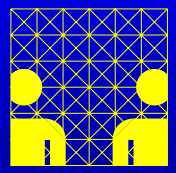
Grundlegende Konzepte

- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...
  - $\Rightarrow$ “Modell” in MVC-Terminologie
  - CLIM: CLOS-Klassen und Methoden (aber nicht zwingend)
- visuelle Repräsentationen der Anwendungsobjekte
  - Display-Objekte $\neq$ Anwendungsobjekte!
  - evtl. verschiedene Darstellungsmöglichkeiten (z.B. Bild des Studenten vs. alphanumerische Daten, “multi-modal”)

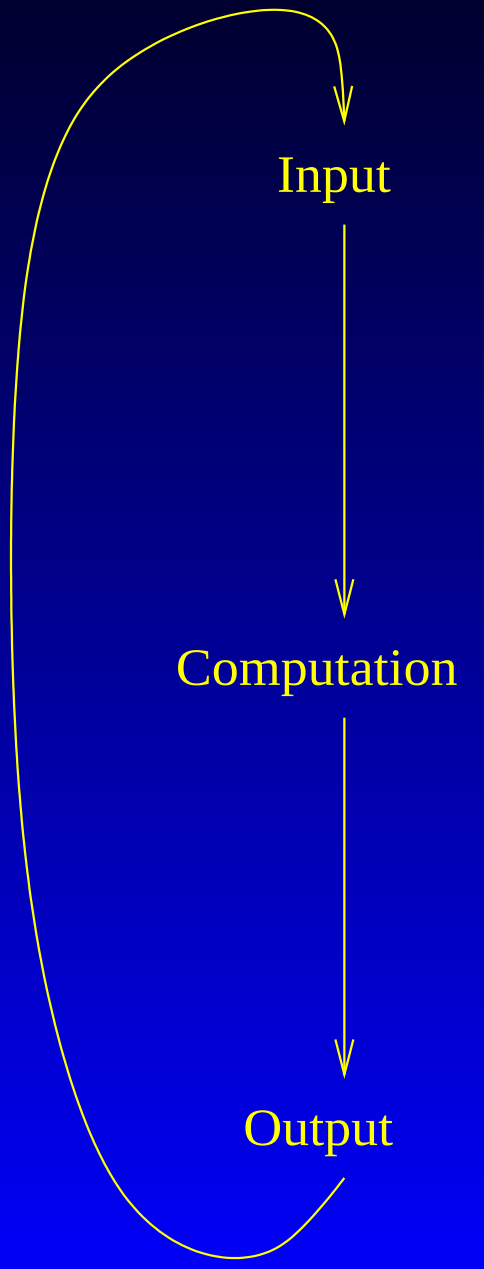


Grundlegende Konzepte

- Anwendungsobjekte
  - z.B. Instanzen der Klassen `person`, `student`, `professor`, ...
  - $\Rightarrow$ “Modell” in MVC-Terminologie
  - CLIM: CLOS-Klassen und Methoden (aber nicht zwingend)
- visuelle Repräsentationen der Anwendungsobjekte
  - Display-Objekte $\neq$ Anwendungsobjekte!
  - evtl. verschiedene Darstellungsmöglichkeiten (z.B. Bild des Studenten vs. alphanumerische Daten, “multi-modal”)
  - $\Rightarrow$ “Views” in MVC-Terminologie



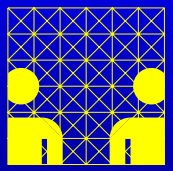
Grundlegende Konzepte



User enters command
Program prompts for and accepts valid arguments

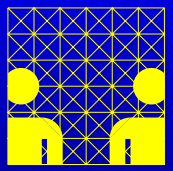
Application objects are created or
reused automatically
Program manipulates its application objects
Display objects are created (or updated)
from application objects
Application and display objects continue to exist
Display objects remain linked
to application objects

User sees display objects
User can operate on display objects



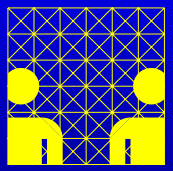
Grundlegende Konzepte (2)

- Display-Objekte und Anwendungsobjekte sind gekoppelt! (Vererbung? Assoziation? . . .)



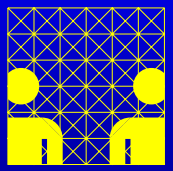
Grundlegende Konzepte (2)

- Display-Objekte und Anwendungsobjekte sind gekoppelt! (Vererbung? Assoziation? . . .)
- Konsistenz zwischen “Modell” und “View”



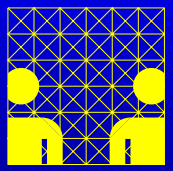
Grundlegende Konzepte (2)

- Display-Objekte und Anwendungsobjekte sind gekoppelt! (Vererbung? Assoziation? . . .)
- Konsistenz zwischen “Modell” und “View”
  - Das Display-Objekt muss den aktuellen Zustand des Anwendungsobjektes korrekt widerspiegeln: $\boxed{\text{Model} \Rightarrow \text{View}}$ (Events? Methodenaufrufe? . . .)



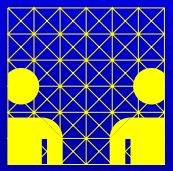
Grundlegende Konzepte (2)

- Display-Objekte und Anwendungsobjekte sind gekoppelt! (Vererbung? Assoziation? . . .)
- Konsistenz zwischen “Modell” und “View”
  - Das Display-Objekt muss den aktuellen Zustand des Anwendungsobjektes korrekt widerspiegeln: $\boxed{\text{Model} \Rightarrow \text{View}}$ (Events? Methodenaufrufe? . . .)
  - Der Benutzer führt Kommandos (auch direktmanipulativ auf den Display-Objekten bzw. Views) aus, die den Zustand der Anwendung verändern sollen: $\boxed{\text{View} \Rightarrow_{\text{Controller}} \text{Model}}$ (Events? Methodenaufrufe? . . .)



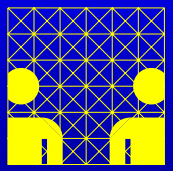
Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”



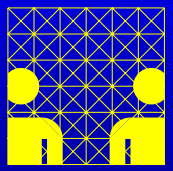
Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!



Display- ⇔ Anwendungsobjekt

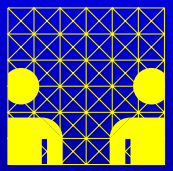
- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)



Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

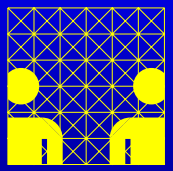


Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

- Typisierter und memorierter Output ermöglicht

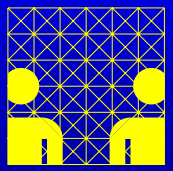


Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

- Typisierter und memorierter Output ermöglicht
  - Kontextsensitivität von Kommandos

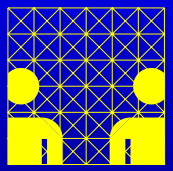


Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

- Typisierter und memorierter Output ermöglicht
  - Kontextsensitivität von Kommandos
  - automatisches Highlighting

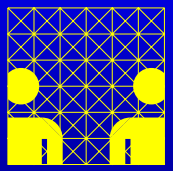


Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

- Typisierter und memorierter Output ermöglicht
  - Kontextsensitivität von Kommandos
  - automatisches Highlighting
  - kontextsensitive Hilfe und Vervollständigung!



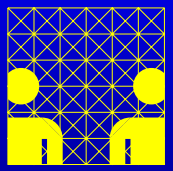
Display- $\Leftrightarrow$ Anwendungsobjekt

- Konzept der “Presentations”
- Geniale Idee Nr. 1: Output ist in CLIM typisiert und wird “nicht vergessen”!
- Presentation= (Anwendungsobjekt, PType, Output Record)

$\Rightarrow$ Presentation Types (PTypes)

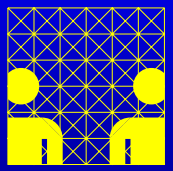
- Typisierter und memorierter Output ermöglicht
  - Kontextsensitivität von Kommandos
  - automatisches Highlighting
  - kontextsensitive Hilfe und Vervollständigung!

  $\Rightarrow$ “intelligentes” UI!



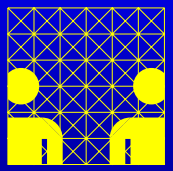
Übergang Model ⇒ View

- `present`-Methoden



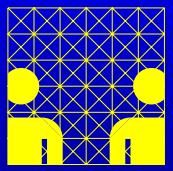
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”



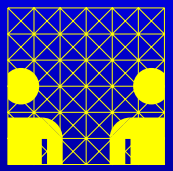
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)



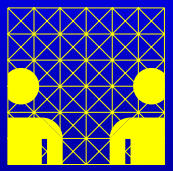
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)



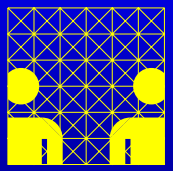
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)
- Präsentation mittels `present` erzeugt “Presentations”



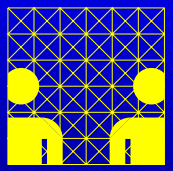
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)
- Präsentation mittels `present` erzeugt “Presentations”
- Output wird memoriert in “Output Records”



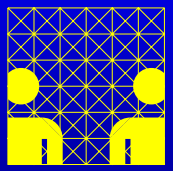
Übergang Model ⇒ View

- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)
- Präsentation mittels `present` erzeugt “Presentations”
- Output wird memoriert in “Output Records”
- Multi-Modalität durch Angabe eines “Views”



Übergang Model ⇒ View

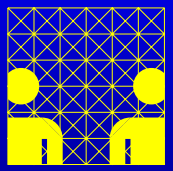
- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)
- Präsentation mittels `present` erzeugt “Presentations”
- Output wird memoriert in “Output Records”
- Multi-Modalität durch Angabe eines “Views”
- Protokoll f. inkrementelles Redisplay



Übergang Model ⇒ View

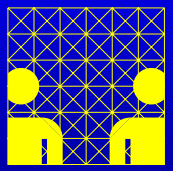
- `present`-Methoden
- Konzept der “Sheets”
  - Sämtliche Output-Operationen erfolgen auf abstrakten Darstellungs-Medien (Sheets)
  - physische Darstellung wird durch “Rendering” erzeugt (Schirm, Postscript-Stream, . . .)
- Präsentation mittels `present` erzeugt “Presentations”
- Output wird memoriert in “Output Records”
- Multi-Modalität durch Angabe eines “Views”
- Protokoll f. inkrementelles Redisplay

  ⇒ “Cache Values”



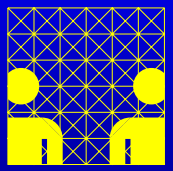
Übergang View ⇒ Model

- Geniale Idee Nr. 2: CLIM “Commands”



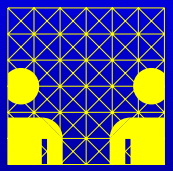
Übergang View ⇒ Model

- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal



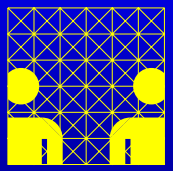
Übergang View ⇒ Model

- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal
  - textuell per Kommandozeile, aus dem Menü, per Knopfdruck, direkt-manipulativ auf den Presentations, . . .



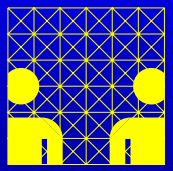
Übergang View ⇒ Model

- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal
  - textuell per Kommandozeile, aus dem Menü, per Knopfdruck, direkt-manipulativ auf den Presentations, . . .
- Kommandos benötigen Argumente bestimmter Typen:



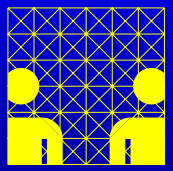
Übergang View ⇒ Model

- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal
  - textuell per Kommandozeile, aus dem Menü, per Knopfdruck, direkt-manipulativ auf den Presentations, . . .
- Kommandos benötigen Argumente bestimmter Typen:
  - Argumente sind Presentations entsp. PTypes



Übergang View ⇒ Model

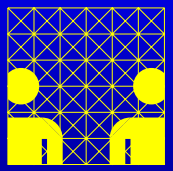
- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal
  - textuell per Kommandozeile, aus dem Menü, per Knopfdruck, direkt-manipulativ auf den Presentations, . . .
- Kommandos benötigen Argumente bestimmter Typen:
  - Argumente sind Presentations entsp. PTypes
  - Commandos etablieren einen Input-Kontext



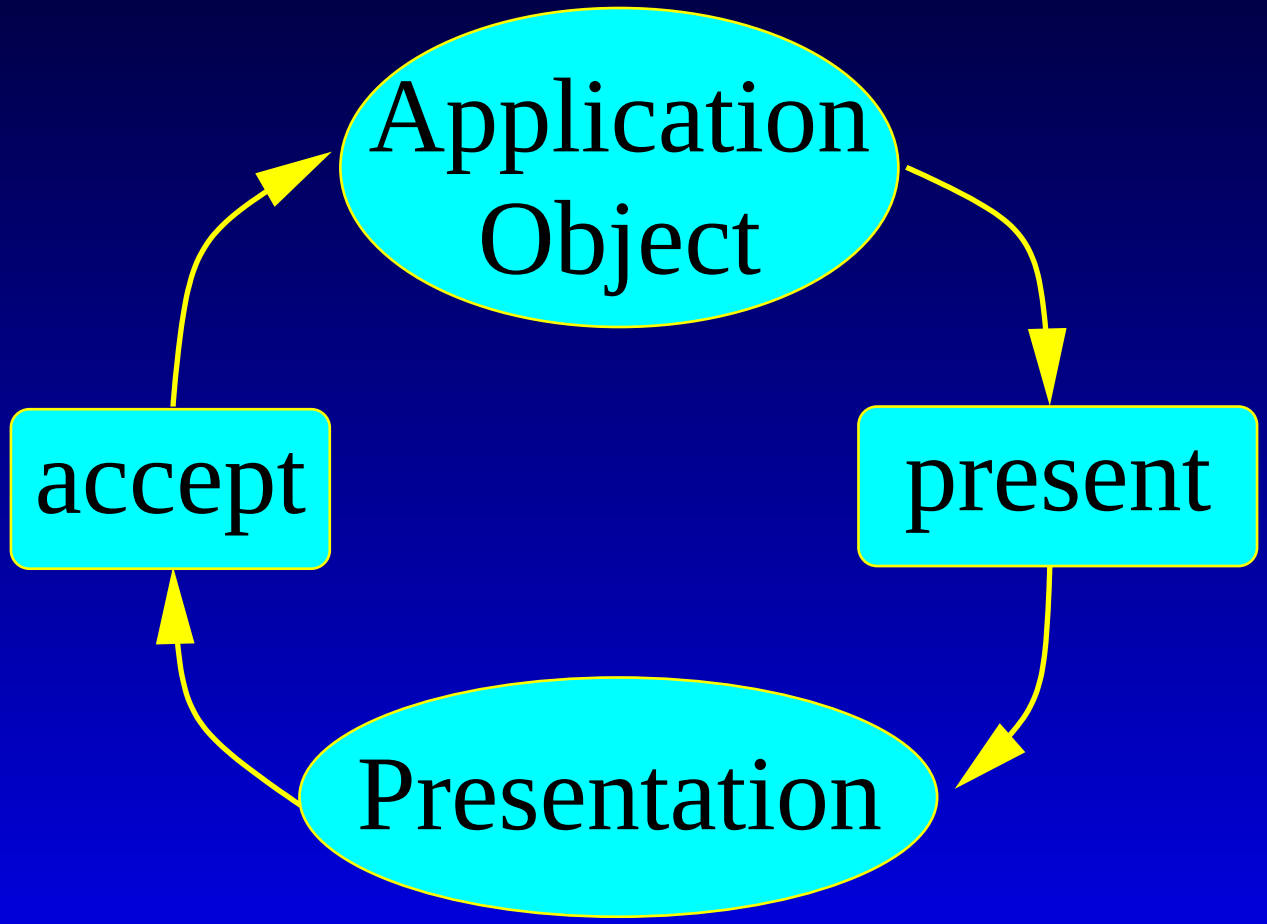
Übergang View ⇒ Model

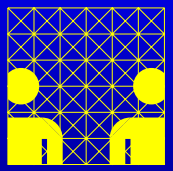
- Geniale Idee Nr. 2: CLIM “Commands”
- Kommandos können auf vielfältige Weise ausgelöst werden, sind multi-modal
  - textuell per Kommandozeile, aus dem Menü, per Knopfdruck, direkt-manipulativ auf den Presentations, . . .
- Kommandos benötigen Argumente bestimmter Typen:
  - Argumente sind Presentations entsp. PTypes
  - Commandos etablieren einen Input-Kontext

  ⇒ alle Presentations des entsp. (Sub)-Typs sind automatisch maussensitiv! Fehleingaben werden gar nicht erst zugelassen!



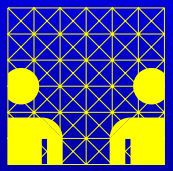
Der `present/accept`-Zyklus





Presentation Translators

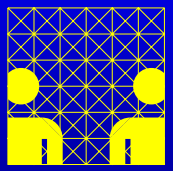
- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!



Presentation Translators

- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!

⇒ `define-presentation-translator source-type ⇒ target-type`

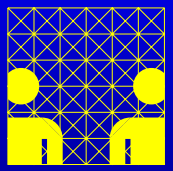


Presentation Translators

- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!

⇒ `define-presentation-translator source-type` ⇒ `target-type`

- Commands sind ebenfalls Presentations!

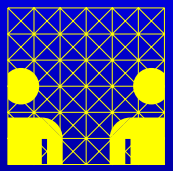


Presentation Translators

- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!

$\Rightarrow$ `define-presentation-translator source-type` $\Rightarrow$ `target-type`

- Commands sind ebenfalls Presentations!
  - Z.B. History-Operationen auf Commands definierbar

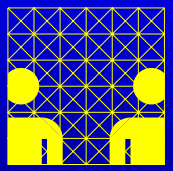


Presentation Translators

- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!

⇒ `define-presentation-translator source-type ⇒ target-type`

- Commands sind ebenfalls Presentations!
  - Z.B. History-Operationen auf Commands definierbar
  - “Meta-Ebene”



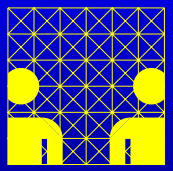
Presentation Translators

- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit “beliebig” (in Abhängigkeit vom Kontext) uminterpretieren zu können!

⇒ `define-presentation-translator source-type ⇒ target-type`

- Commands sind ebenfalls Presentations!
  - Z.B. History-Operationen auf Commands definierbar
  - “Meta-Ebene”

⇒ `define-presentation-to-command-translator`



Presentation Translators

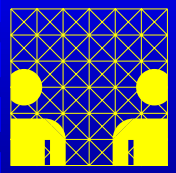
- Geniale Idee Nr. 3: biete Abbildungen an, um den PType einer Presentation zur Laufzeit "beliebig" (in Abhängigkeit vom Kontext) uminterpretieren zu können!

⇒ `define-presentation-translator source-type ⇒ target-type`

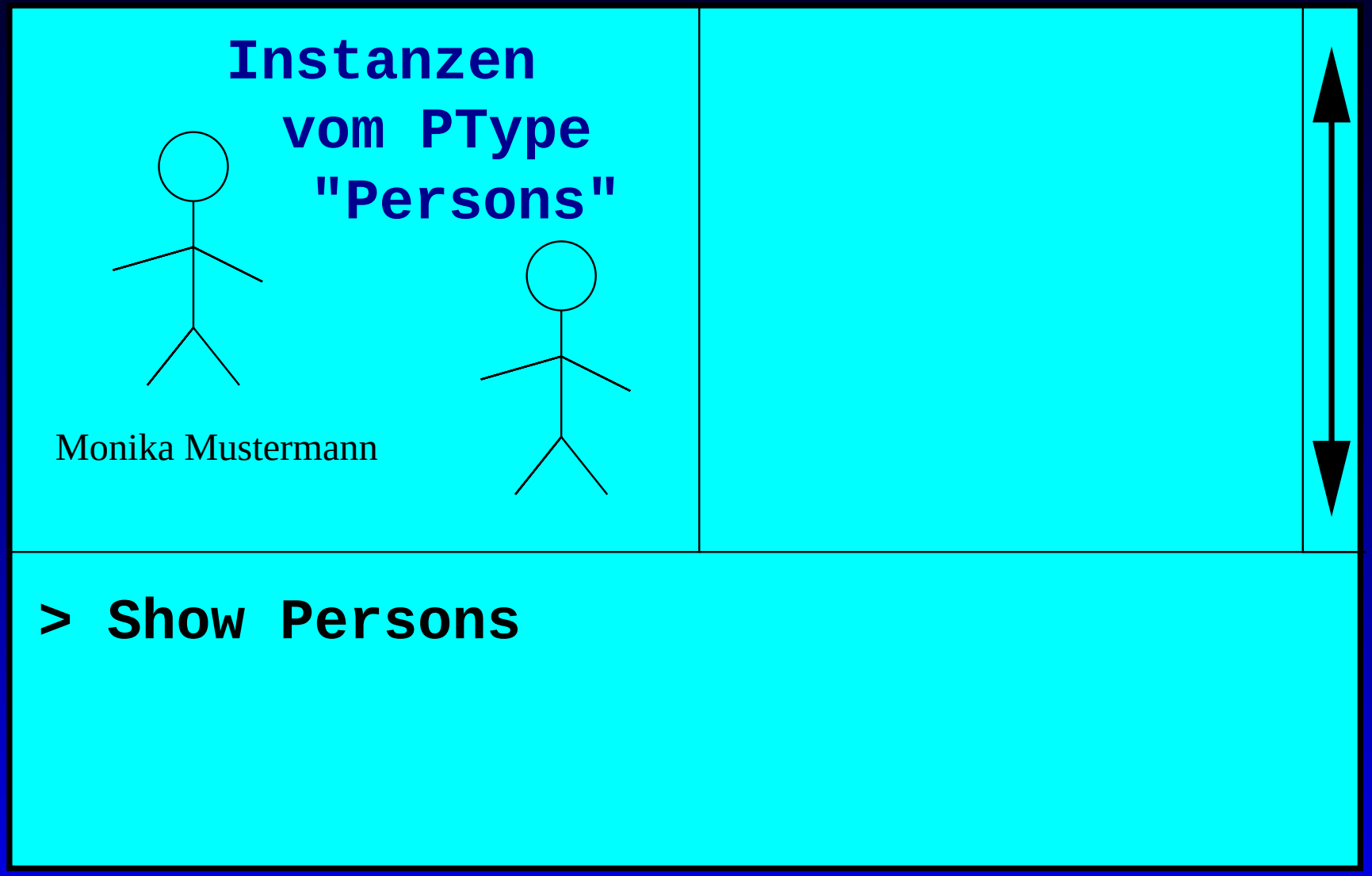
- Commands sind ebenfalls Presentations!
  - Z.B. History-Operationen auf Commands definierbar
  - "Meta-Ebene"

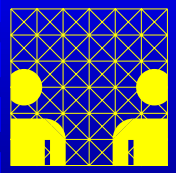
⇒ `define-presentation-to-command-translator`

- z.B. `person` ⇒ `com-show-address`

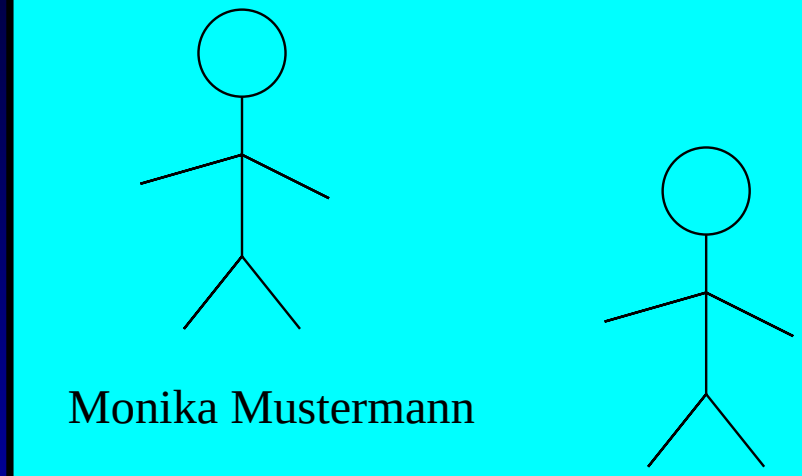


Presentation Translators





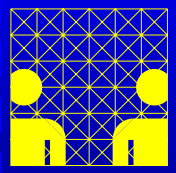
Presentation Translators



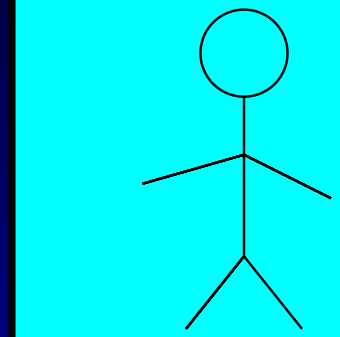

```
Name: Monika Mustermann
Straße: Musterstr. 3
PLZ, Stadt: 8888, Musterhausen
Land: Deutschland
```

```
... Instanzen
...   vom PType
...   "Address"
```

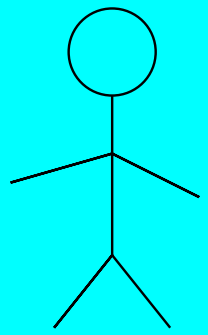
```
> Show Persons
> Show Address List
```



Presentation Translators



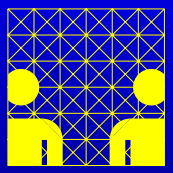
Monika Mustermann



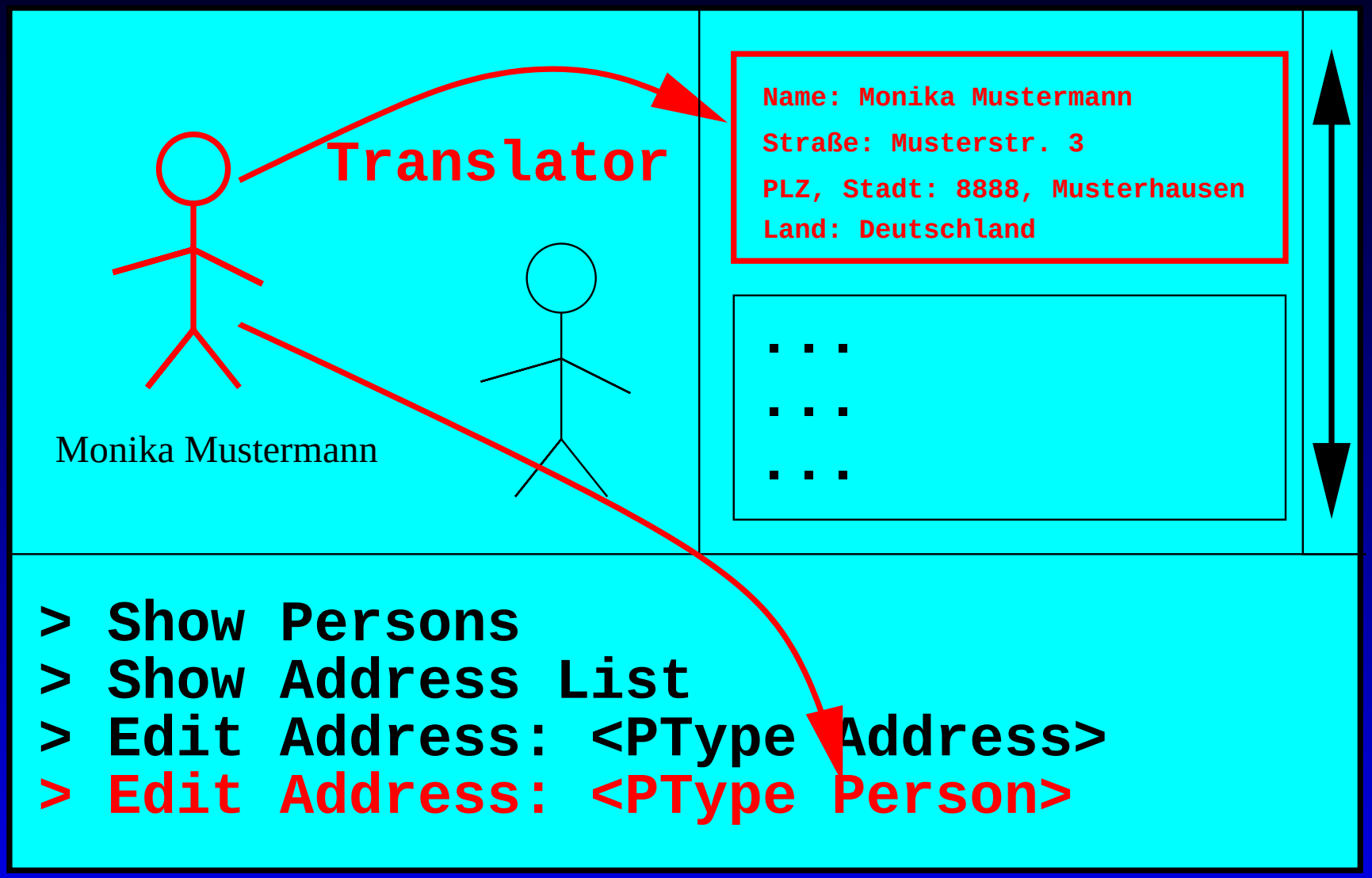
Name: Monika Mustermann
Straße: Musterstr. 3
PLZ, Stadt: 8888, Musterhausen
Land: Deutschland

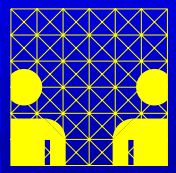
. . .
. . .
. . .

> Show Persons
> Show Address List
> Edit Address: <PType Address>



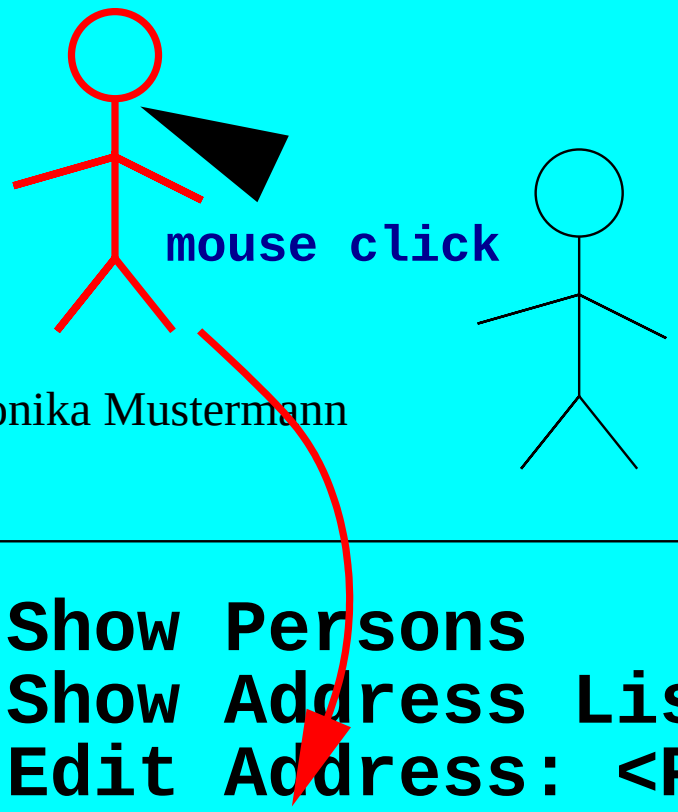
Presentation Translators





Presentation Translators

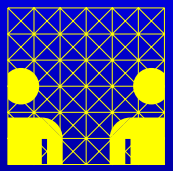
Presentation-to-command-translator



Name: Monika Mustermann
Straße: Musterstr. 3
PLZ, Stadt: 8888, Musterhausen
Land: Deutschland

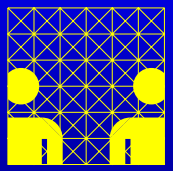
...
...
...

```
> Show Persons
> Show Address List
> Edit Address: <PType Address>
> Show Address: <PType Person>
```



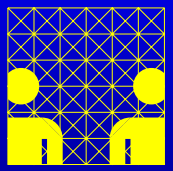
Weitere Features

- Output wird in “Output Records” memoriert



Weitere Features

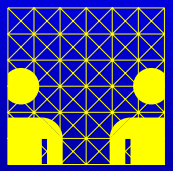
- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)



Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

⇒ Allgemeine Formatierungsfunktionen

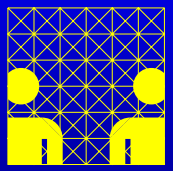


Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

$\Rightarrow$ Allgemeine Formatierungsfunktionen

- Formatierung von Tabellen und Graphen

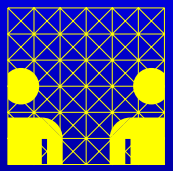


Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

⇒ Allgemeine Formatierungsfunktionen

- Formatierung von Tabellen und Graphen
- Layout-Berechnung durch Constraint-Satisfaction-Verfahren

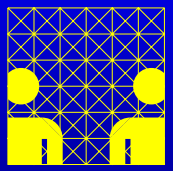


Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

⇒ Allgemeine Formatierungsfunktionen

- Formatierung von Tabellen und Graphen
- Layout-Berechnung durch Constraint-Satisfaction-Verfahren
- Affine Transformationen



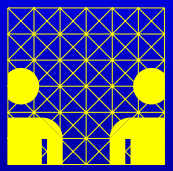
Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

⇒ Allgemeine Formatierungsfunktionen

- Formatierung von Tabellen und Graphen
- Layout-Berechnung durch Constraint-Satisfaction-Verfahren

- Affine Transformationen
- Farbmodelle



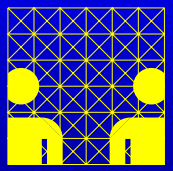
Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

⇒ Allgemeine Formatierungsfunktionen

- Formatierung von Tabellen und Graphen
- Layout-Berechnung durch Constraint-Satisfaction-Verfahren

- Affine Transformationen
- Farbmodelle
- `with-output-to-postscript-stream`

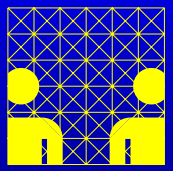


Weitere Features

- Output wird in “Output Records” memoriert
- Output kann beliebig repositioniert und neu “abgespielt” werden (Scrolling, dynamisches Highlighting, . . .)

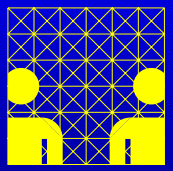
⇒ Allgemeine Formatierungsfunktionen

  - Formatierung von Tabellen und Graphen
  - Layout-Berechnung durch Constraint-Satisfaction-Verfahren
- Affine Transformationen
- Farbmodelle
- `with-output-to-postscript-stream`
- . . .



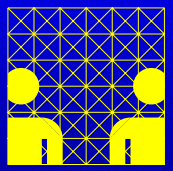
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz



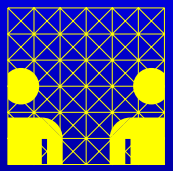
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen



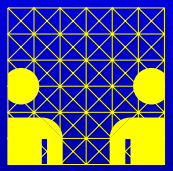
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`



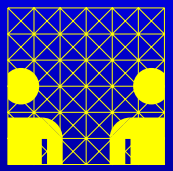
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`
  - `place, transition, place-with-tokens, place-with-capacity, edge, edge-with-capacity`



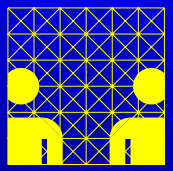
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`
  - `place, transition, place-with-tokens, place-with-capacity, edge, edge-with-capacity`
- CLOS-Methoden



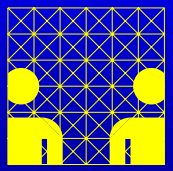
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`
  - `place, transition, place-with-tokens, place-with-capacity, edge, edge-with-capacity`
- CLOS-Methoden
  - `create-place/transition`



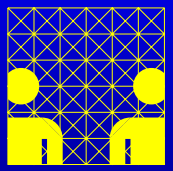
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`
  - `place, transition, place-with-tokens, place-with-capacity, edge, edge-with-capacity`
- CLOS-Methoden
  - `create-place/transition`
  - `remove-from-net, link, unlink, add-tokens, remove-tokens, add-capacity-label`



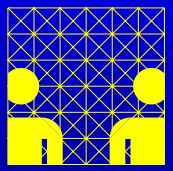
Beispiel: Petri-Netz-Editor

- “ADT” Petri-Netz
- CLOS-Klassen
  - `petri-net, token-net`
  - `place, transition, place-with-tokens, place-with-capacity, edge, edge-with-capacity`
- CLOS-Methoden
  - `create-place/transition`
  - `remove-from-net, link, unlink, add-tokens, remove-tokens, add-capacity-label`
  - `activated-p, step-net`



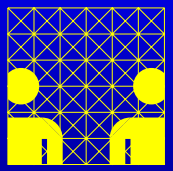
Views für den Editor

- Kopplung an Anwendungsobjekte durch Mehrfachvererbung



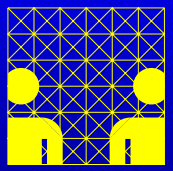
Views für den Editor

- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute



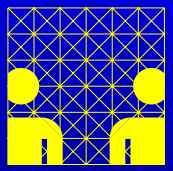
Views für den Editor

- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute
  - `place-view` $\sqsubseteq$ `positioned-display-object` $\wedge$ `place`



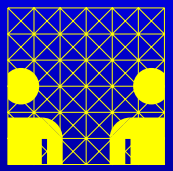
Views für den Editor

- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute
  - `place-view` $\sqsubseteq$ `positioned-display-object` $\wedge$ `place`
  - die Kopplung ist lose genug, da nur über das Methoden-Protokoll auf das Modell zugegriffen wird!



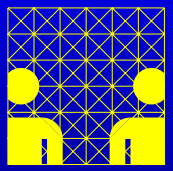
Views für den Editor

- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute
  - `place-view` $\sqsubseteq$ `positioned-display-object` $\wedge$ `place`
  - die Kopplung ist lose genug, da nur über das Methoden-Protokoll auf das Modell zugegriffen wird!
  - Alternativ: Kopplung per Assoziation (oftmals “loser”)



Views für den Editor

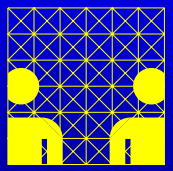
- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute
  - `place-view` $\sqsubseteq$ `positioned-display-object` $\wedge$ `place`
  - die Kopplung ist lose genug, da nur über das Methoden-Protokoll auf das Modell zugegriffen wird!
  - Alternativ: Kopplung per Assoziation (oftmals “loser”)
- Problem: wie bringe ich das Petri-Netz-Modell dazu, entsp. Display-Objekte zu erzeugen? (aka “`make` isn’t generic problem”)



Views für den Editor

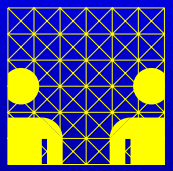
- Kopplung an Anwendungsobjekte durch Mehrfachvererbung
  - Views benötigen zusätzliche Attribute
  - `place-view` $\sqsubseteq$ `positioned-display-object` $\wedge$ `place`
  - die Kopplung ist lose genug, da nur über das Methoden-Protokoll auf das Modell zugegriffen wird!
  - Alternativ: Kopplung per Assoziation (oftmals “loser”)
- Problem: wie bringe ich das Petri-Netz-Modell dazu, entsp. Display-Objekte zu erzeugen? (aka “`make` isn’t generic problem”)

  $\Rightarrow$ “Fabrik-Erzeugungsmuster”



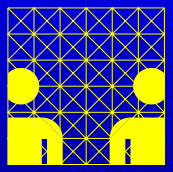
CLOS

```
(defclass a () ()) ; Klasse a
(defclass b (a) ()) ; Unterkl. von a
(defclass c (b) ()) ; Unterkl. von b

(defmethod test ((a a))
  ;; primäre Methode f. a
  (princ "prim-a") (terpri))

(defmethod test ((b b))
  (princ "prim-b") (terpri))

(defmethod test ((c c))
  (princ "prim-c") (terpri))
```



CLOS (2)

```
(defmethod test :before ((a a))
  (princ "before-a") (terpri))
(defmethod test :before ((b b))
  (princ "before-b") (terpri))
(defmethod test :before ((c c))
  (princ "before-c") (terpri))

(defmethod test :after ((a a))
  (princ "after-a") (terpri))
(defmethod test :after ((b b))
  (princ "after-b") (terpri))
(defmethod test :after ((c c))
  (princ "after-c") (terpri))
```



CLOS (3)

```
(defmethod test :around ((a a))
  (princ "around-a") (terpri)
  (call-next-method))

(defmethod test :around ((b b)) ...)

(defmethod test :around ((c c)) ...)

(test (make-instance 'c))

;;; around-c around-b around-a
;;; before-c before-b before-a
;;; prim-c
;;; after-a after-b after-c
```