# Kurz-Tutorial ObjectGEODE

Helmut Neukirchen
neukirchen@itm.mu-luebeck.de
Institut für Telematik
Medizinische Universität zu Lübeck

19. Juni 2000

## 1 Vorbereitungen

In der `.login`-Datei muß der Pfad für ObjectGEODE gesetzt sein.

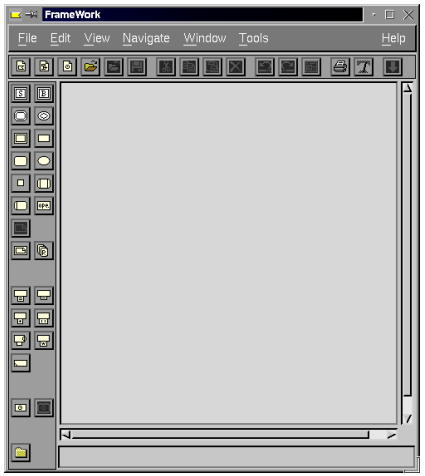
Es empfiehlt sich, für die im Laufe des Tutorials erzeugten Dateien, ein eigenes Verzeichnis anzulegen (z.B. `SDL_Tutorial`) und in dieses zu wechseln.

Außerdem muß ein C-Compiler installiert sein. Der SUN SPARCompiler tut's auf jeden Fall – der GCC könnte es aber auch tuen…

## 2 Starten

ObjectGEODE wird mit `og` gestartet. Neben einem Fenster mit der Versionsnummer, das nach einiger Zeit von selbst wieder verschwindet, erscheint das *FrameWork*-Fenster.

`og`




Da noch keine Dateien bearbeitet wurden, ist die Arbeitsfläche noch leer…



## 3 Editieren

### 3.1 Ein Projekt anlegen

Zunächst muß ein "Projekt" angelegt werden: Aus dem Submenü *New* des *File*-Menüs den Punkt *Project* auswählen. Es erscheint ein Eintrag für das neue Projekt. Einen vernünftigen Namen bekommt es, indem es mit *Save As …* aus dem Menü *File* abgespeichert wird. Als neuen Dateinamen `Cola_Automat` eingeben und *OK* klicken.

*File→New→Project*

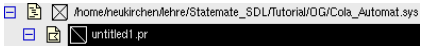

*File→Save As …*
`Cola_Automat`
*OK*

(Falls solch ein "Projekt" später per *Load* wieder geladen werden soll, muß in der Dateiauswahl auf *Project Files *.sys* umgestellt werden.)

Dem angezeigten Projekt kann nun der SDL-Teil hinzugefügt werden: Die Zeile mit dem Projekt selektieren und aus dem Submenü *New* des *File*-Menüs den Punkt *SDL* auswählen.

Zeile des Projekts selektieren
*File→New→SDL*

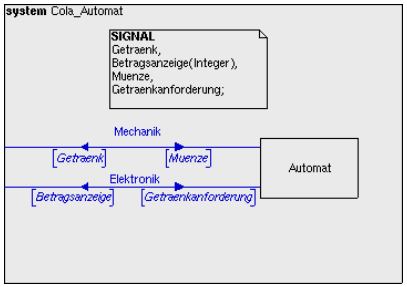
Eine neue Zeile erscheint unterhalb der Projektzeile. Die Einrückung deutet an, daß dieser Teil zu dem darüberliegenden Projekt gehört.




Per *Save As …* aus dem Menü *File* kann dem SDL-Teil ebenfalls ein neuer Name gegeben werden: `Cola_Automat`

*File→Save As …*
`Cola_Automat`, *OK*

Im folgenden soll ein einfacher Cola-Automat, der mit Münzen gefüttert wird und auf Tastendruck ein Kaltgetränk ausgibt (sofern der eingeworfene Geldbetrag, der angezeigt wird, ausreicht) in SDL modelliert werden:




### 3.2 Ein SDL-System hinzufügen

Das eigentliche, oben abgebildete *System*-Diagramm wird zum Projekt hinzugefügt, indem die Zeile *Cola_Automat.pr* selektiert und anschließend auf das Icon *System* in der linken Symbolleiste geklickt wird. (Die Namen der Icons werden angezeigt, sobald man sich mit dem Mauszeiger darüber befindet.)

*Cola_Automat.pr* selektieren

*System* anklicken

Eine weitere Zeile, die ein SDL-Diagramm vom Typ *System* repräsentiert, erscheint. Dieses *System* muß noch einen Namen bekommen. Hierzu wird am Besten – die gerade neu erstellte *System*-Zeile muß noch selektiert sein – das Icon *Content* aus der oberen Symbolleiste verwendet. Daraufhin erscheint ein Fenster *SDL-Content*, in dem der Name des *Systems* eingeben wird: `Cola_Automat`. Dieser Dialog wird mit *OK* bestätigt.

*Content* anklicken

`Cola_Automat`, *OK*

## 3.3 System editieren

Das zu Beginn abgebildete SDL-*System Cola_Automat* soll nun bearbeitet werden: Im *FrameWork*-Fenster auf die soeben erstellte Zeile mit dem *System Cola_Automat* doppelklicken. Nun wird der ein SDL-Editor-Fenster *Interconnection* geöffnet. Dieser zeigt ein leeres *System Cola_Automat* an. In den nächsten Abschnitten wird ausschließlich in diesem Fenster gearbeitet.

Doppelklick auf *Cola_Automat*

### 3.3.1 Block hinzufügen

Als erstes soll ein SDL-*Block* hinzugefügt werden. Hierzu aus der linken Symbolleiste das Icon *block* mit der linken Maustaste kurz anklicken (innerhalb der Arbeitsfläche verwandelt sich der Mauszeiger nun in ein Kreuz) und irgendwo in der Mitte der Arbeitsfläche per Mausklick plazieren.

*block* anklicken

plazieren

Das *Block*-Symbol erscheint mit einem dicken Cursor in der Mitte. Dieser wartet darauf, daß diesem *Block* ein Name gegeben wird: `Automat`. Ein Klick außerhalb des Textfeldes beendet die Eingabe.

`Automat` eingeben

Klick außerhalb

### 3.3.2 Kanal hinzufügen

Der *Block Automat* soll eine Verbindung (*Kanal* mit dem *Environment* (der Außenwelt, die durch den rechteckigen Rahmen um das *System* herum begrenzt ist) bekommen. Hierzu wird das *link*-Icon verwendet. Nach dem Anklicken dieses Icons sind der Start- und der Endpunkt des *Kanals* festzulegen:

Der Startpunkt wird durch kurzen Klick mit der linken Maustaste am z.B. oberen, linken Rand des *Block*-Symbols festgelegt. Der Endpunkt wird z.B. am linken Rand des *Environments* festgelegt, indem dort erneut die linke Maustaste gedrückt wird.

Mausklick am oberen, linken Rand des *Block*-Symbols

Mausklick am oberen, linken Rand des *Environments*

Es erscheinen daraufhin drei neue Textfelder unterhalb des *Kanals*. Zunächst muß der Name des Kanals eingeben werden: `Mechanik` eingeben und auf den linken Pfeil klicken. Dort werden die Namen der *Signale*, die über diesen *Kanal* zum *Environment* geschickt werden, eingegeben: `Getraenk`. Auf den rechten Pfeil klicken und anschließend `Muenze` eingeben, um den Signalfluß in der Gegenrichtung zu spezifizieren.


`Mechanik` eingeben

auf den linken Pfeil klicken

`Getraenk` eingeben

auf den rechten Pfeil klicken, `Muenze` eingeben



Nach einem Klick irgendwo auf den leeren Arbeitsbereich können noch optische Schönheitskorrekturen vorgenommen wurden. So empfiehlt es sich, den Kanalnamen oberhalb des *Kanals* zu positionieren. Die *Kanäle* selber können mit der Maus verschoben werden; Textfelder können “angepackt” (die Mauszeigerform ändert sich) und verschoben werden. Textfelder können nachträglich editiert oder per Eingabetaste umgebrochen werden.

Analog muß ein zweiter *Kanal Elektronik*, der – wie nebenstehend abgebildet – parallel zum *Environment* verläuft, erstellt werden. Zum *Environment* läuft das *Signal Betragsanzeige*, in der Gegenrichtung das *Signal Getraenkanforderung*.

Zweiten *Kanal* `Elektronik` anlegen, *Signale*: `Betragsanzeige` und `Getraenkanforderung`


### 3.3.3 Signaldeklaration

Die ausgetauschten *Signale* müssen noch genauer bzgl. ihres Typs deklariert werden. Dies geschieht innerhalb eines *Text*-Symbols:

Aus der Symbolleiste *text* auswählen und z.B. in die Mitte der oberen Arbeitsfläche plazieren. (Intuitiv landet dieses Symbol gerne ein wenig außerhalb der Arbeitsfläche, aber das kann hinterher immer noch verschoben werden!)

*text* auswählen

plazieren

In diesem Symbol wartet ebenfalls ein Cursor und die Signaldeklaration kann eingegeben werden:

```
SIGNAL Getraenk, Betragsanzeige(Integer),
Muenze, Getraenkanforderung;
```

```
SIGNAL Getraenk, Betragsanzeige(Integer), Muenze, Getraenkanforderung; eingeben
```

Ein Klick “in’s Leere” beendet die Eingabe und ermöglicht es z.B. das Symbol noch ein wenig zu verschieben.

“in’s Leere” klicken

Das eingegebene *System* sollte nun dem in Abschnitt 3.1 gezeigten SDL *System Cola_Automat* ähneln.

### 3.3.4 Speichern

Durch Auswahl von *Save* im Menüpunkt *File* und Bestätigen mit *OK* kann das *System* gespeichert werden.

*File→Save*

*OK*

Mit *Close* im Menüpunkt *Window* kann der Editor beendet werden. (*Exit* hingegen beendet das komplette ObjectGEODE-Tool!)

*Window→Close*

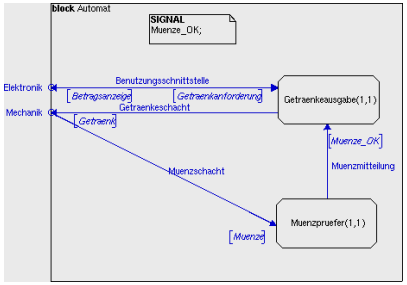
Im *FrameWork*-Fenster ist nun unter dem *System Cola_Automat* ein neuer Eintrag hinzugekommen, der den neu angelegten *Block Automat* repräsentiert .

## 3.4 Einen SDL-Block editieren

Nun soll der *Block Automat* editiert werden: Ein Doppelklick auf die Zeile *Automat* im *FrameWork*-Fenster öffnet das bekannte *Interconnection*-Fenster mit dem *Block Automat*.

Doppelklick auf *Automat* im *FrameWork*

In diesem Abschnitt soll der nachfolgende SDL-*Block* erstellt werden:




### 3.4.1 Prozess hinzufügen

Zunächst wird das Icon *process* ausgewählt und in die obere Hälfte der Arbeitsfläche platziert. Dieser neue *Prozess* bekommt den Namen `Getraenkeausgabe(1,1)`. Die Angabe `(1,1)` legt fest, daß von diesem *Prozess* genau 1 Instanz angelegt werden kann.

*process* auswählen und plazieren
`Getraenkeausgabe(1,1)`

Der zweite *Prozess* soll `Muenzpruefer(1,1)` heißen und kann auf die gleiche Weise darunter plaziert werden.

*Prozess* `Muenzpruefer(1,1)` anlegen und plazieren

### 3.4.2 Signalrouten hinzufügen

*Prozesse* besitzen – wie die *Blöcke* – Verbindungen untereinander und zum *Environment*. Auf dieser Ebene heißen sie jedoch *Signalrouten* und stellen eine Verfeinerung von *Kanälen* dar.

Die erste *Signalroute* soll von der *Getraenkeausgabe* zum *Environment* gehen: Hierzu wird – wie für die *Kanäle* – das *link*-Icon verwendet. Nach dem Anklicken dieses Icons sind entsprechend der Start- und der Endpunkt der *Signalroute* festzulegen:

Der Startpunkt wird durch kurzen Klick mit der linken Maustaste am z.B. oberen, linken Rand von *Getraenkeausgabe* festgelegt. Der Endpunkt wird z.B. am linken Rand des *Environments* festgelegt, indem dort erneut die linke Maustaste gedrückt wird.

Mausklick am oberen, linken Rand des *Environments*

Es erscheinen daraufhin die neue *Signalroute* mit drei neuen Textfeldern und ein Dialogfenster *Connections*, das Tipparbeit spart: Hierzu muß zunächst der *Kanal*, der durch diese *Signalroute* verfeinert wird, angegeben werden. Ein Doppelklick auf

Doppelklick auf *Elektronik*



*Elektronik* erledigt dies. Am *Environment* wird nun der *Kanalname* angezeigt und in dem Dialogfenster stehen nun die möglichen *Signale*, die über diesen *Kanal* transportiert werden, zur Auswahl.

(Falls später noch einmal das Dialogfenster *Connections* aufgerufen werden soll, muß der *connection endpoint* des zugehörigen *Kanal*namens leicht außerhalb der Umrandungslinie des *Environments* selektiert werden. Im Kontextmenü, das nach Drücken der rechten Maustaste erscheint, kann dann *Attributes* … ausgewählt werden, was zum gewünschten Dialog führt.)

Da in beiden Richtungen die jeweils zur Verfügung stehenden *Signale* verschickt werden sollen, kann guten Gewissens auf beide *Signale* doppelt geklickt werden. *Close* beendet die Doppelklick-Orgie.

Doppelklick auf *Betragsanzeige* und *Getraenkanforderung*
*Close*

Was noch fehlt, ist der Name dieser *Signalroute*. Hierzu muß mit der linken Maustaste auf die Linie der *Signalroute* geklickt werden. Diese wird daraufhin selektiert und ein Balkencursor erscheint, so daß der zugehörige Name eingetippt werden kann: `Benutzungsschnittstelle`. (Auch hier empfiehlt es sich, diesen Namen nachträglich oberhalb der Linie zu positionieren, um zu verhindern, daß sich die unterschiedlichen Namen überdecken.)

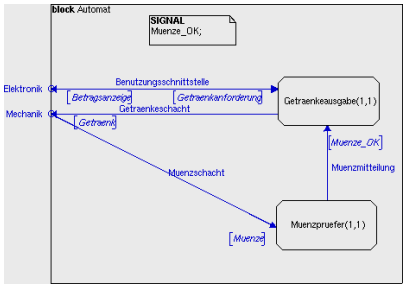
Linie der *Signalroute* anklicken

`Benutzungsschnittstelle`

Analog dazu müssen die verbleibenden *Signalrouten Getraenkeschacht*, *Muenzschacht* und *Muenzmitteilung* wie unten abgebildet angelegt werden.

*Signalrouten Getraenkeschacht*, *Muenzschacht* und *Muenzmitteilung* analog anlegen

Dabei ist zu beachten, daß die verbleibenden *Signalrouten* nur in eine Richtung gerichtet sind. Um dies zu erreichen, ist bei selektierter *Signalroute* die rechte Maustaste zu drücken, so daß ein Kontextmenü erscheint. Der Menüpunkt *Attributes* … führt in diesem Fall zu dem Dialogfenster *Channel or Signal Route*. Dort kann mit den ersten beiden Punkten die Richtung (*From ENV*/*To ENV*) ausgewählt werden.



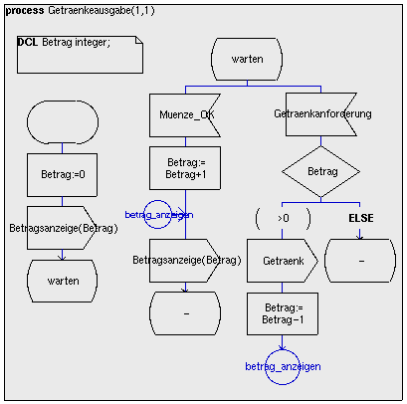
## 3.5 Einen SDL-Prozess editieren

Da die strukturellen Eigenschaften des Automaten nun festgelegt sind, soll nun dessen Verhalten spezifiziert werden. Durch das Hinzufügen der beiden *Prozesse Getraenkeausgabe* und *Muenzpruefer* sind im *FrameWork*-Fenster die zugehörigen Einträge ebenfalls aufgeführt.

Ein Doppelklick auf *Getraenkeausgabe* öffnet das *FSM*-Editor-Fenster für den *Prozess Getraenkeausgabe*.

Doppelklick auf *Getraenkeausgabe* im *FrameWork*

In diesem Abschnitt soll zunächst der nachfolgende SDL-*Prozess* erstellt werden:




### 3.5.1 Flußdiagramm hinzufügen

Die Flußdiagramme innerhalb von *Prozessen* besitzen ein *Start*-Symbol für den Startzustand des SDL-Automaten: *start*-Icon mit der linken Maustaste aus der Symbolleiste rechts auswählen und im linken, oberen Bereich der Arbeitsfläche per Mausklick plazieren. Einen Namen bekommt dieser Zustand nicht, deshalb wird gleich das nächste Symbol angehängt:

*start* auswählen

plazieren

Ohne zwischenzeitlich mit der Maus herumzuklicken, auf das *task*-Icon klicken. Dieses Symbol wird automatisch an das zuletzt selektierte Symbol angehängt. Nun kann der Inhalt dieses *Tasks* eingegeben werden, wobei darauf zu achten ist, daß der Mauszeiger sich innerhalb des Arbeitsbereichs befindet, da sonst die Eingabe ignoriert wird: `Betrag:=0`.

Klick auf *task*

Maus in den Arbeitsbereich bewegen, `Betrag:=0`

Nun soll noch der Betrag ausgegeben werden: bei noch selektiertem *Task*-Symbol auf das *output*-Icon klicken und anschließend den Mauszeiger in den Arbeitsbereich bewegen um den folgenden Text einzugeben: `Betragsanzeige(Betrag)`

Klick auf *output*

Maus in den Arbeitsbereich bewegen, `Betragsanzeige(Betrag)`



Da diese *Transition* (d.h. Baum von Symbolen mit *State*-Symbol an der Wurzel und an den Blättern) jetzt zu Ende sein soll, einfach auf das Icon *nextstate* klicken und – mit Mauszeiger im Arbeitsbereich – den Namen des Folgezustands eingeben: `warten`

*nextstate*

`warten`

Die soeben eingeführte Variable *Betrag* muß noch deklariert werden. Dies geschieht – wie bei der Signaldeklaration – innerhalb eines *Text*-Symbols:

Aus der Symbolleiste *text* auswählen und z.B. in die Mitte der oberen Arbeitsfläche plazieren. (Intuitiv landet dieses Symbol gerne ein wenig außerhalb der Arbeitsfläche, aber das kann hinterher immer noch verschoben werden!)

*text* auswählen

plazieren

In diesem Symbol wartet ein Cursor und die Deklaration kann eingegeben werden:
`DCL Betrag integer;`

`DCL Betrag integer;` eingeben

Ein Klick “in’s Leere” beendet die Eingabe und ermöglicht es z.B. das Symbol noch ein wenig zu verschieben oder zu verkleinern.

“in’s Leere” klicken

Für die nächste *Transition* auf das *state*-Icon in der Symbolleiste klicken und im oberen, rechten Bereich der Arbeitsfläche positionieren. Über Tastatur kann der Namen dieses Zustands eingegeben werden: `warten`. (Wie immer muß der Mauszeiger sich dabei in der Arbeitsfläche befinden.)

*State*

plazieren

`warten`

An das selektierte *State*-Symbol sollen nun – links und rechts – zwei *Input*-Symbole angehängt werden: Zwei einzelne Klicks auf das zugehörige Icon sollten dies zur Folge haben. Es soll auf das *Signal Muenze* gewartet werden. Bei in diesem Symbol blinkendem Cursor, `Muenze` eingeben.

Zwei einzelne Klicks auf *input*

`Muenze`

Um den linken Ast weiter zu bearbeiten, das linke *Input*-Symbol anklicken und den Namen des Symbols, auf dessen Eingabe gewartet wird, eingeben: `Muenze_OK`

linkes *Input*-Symbol anklicken, `Muenze_OK`

Wie der Task `Betrag:=Betrag+1` angelegt wird, sollte ja mittlerweile bekannt sein. . .

*Task* `Betrag:=Betrag+1` anlegen

Die beiden lustigen Kreise symbolisieren ein “Label” bzw. das zugehörige “Goto”: *connection / label* erzeugt ein Label, dem der Name `betrag_anzeigen` gegeben wird.

`betrag_anzeigen`

Das *Output*-Symbol für `Betragsanzeige(Betrag)` sollte auch kein Problem sein. . .

*Output*, `Betragsanzeige(Betrag)`

Im Anschluß soll wieder in den Ausgangszustand übergegangen werden. Da wir diesmal schreibfaul sind, wird nach dem Klick auf das *nextstate*-Icon in der Symbolleiste einfach nur `-` eingegeben, was eine Abkürzung für den Zustand ist, mit dem die *Transition* begonnen hat.

*nextstate*

`-`

Um den rechten Ast weiter zu bearbeiten, das rechte *Input*-Symbol anklicken und den Namen des Symbols, auf dessen Eingabe gewartet wird, eingeben: `Getraenkanforderung`

Nun muß ein *decision*-Symbol plaziert und den Variablennamen `Betrag` eingegeben werden.

Die beiden Äste dieser Verzweigung werden ähnlich wie die beiden *Input*-Symbole angelegt: zweimal hintereinander auf das *answer*-Icon klicken.

Für den *ELSE*-Zweig einfach `ELSE` eingeben und direkt im Anschluß auf das *nextstate*-Icon klicken und einfach nur `-` eingegeben.

Nun den anderen Zweig anklicken und `>0` eingeben.

Für den Rest dieses Zweigs besteht aus der Ausgabe des *Signals Getraenk* und dem *Task Betrag:=Betrag-1*, was leicht einzugeben sein sollte…

Zuletzt muß noch ein “Goto” angehängt werden. Dies geschieht über das Icon *join*. Um das Ziel dieses Sprungs angeben zu können, muß dieses Symbol selektiert sein. Tippfaule benutzen anschließend nicht die Tastatur sondern das *Dictionary*, dessen Dialogfenster sich im *View*-Menü versteckt.

Dort in linken Spalte den Eintrag *Label* suchen und selektieren. In der rechten Spalte sollte daraufhin der Eintrag *betrag_anzeigen* erscheinen, der durch Doppelklick in das *Join*-Symbol eingetragen werden. *Close* schließt das *Dictionary*-Fenster.

Fertig ist das Diagramm!

Das Diagramm für *Getraenkeausgabe* kriegt ihr jetzt doch alleine hin, oder?

Um abschließend das komplette SDL-System per *FrameWork*-Fenster zu speichern ist darauf zu achten, daß irgendeine der Zeilen des SDL-Systems selektiert ist, da sonst der Menüpunkt *Save* erst gar nicht aufgerufen werden kann.

Die evtl. noch offenen Editor-Fenster können nun endgültig geschlossen werden.

rechtes *Input*-Symbol anklicken, `Getraenkanforderung`

*decision*, `Betrag`

zweimal *answer*

`ELSE`

*nextstate*
`-`
`>0`
*output*, `Getraenk`
*task*, `Betrag:=Betrag-1`

*join*
*join*-Symbol selektieren
*View→Dictionary …*

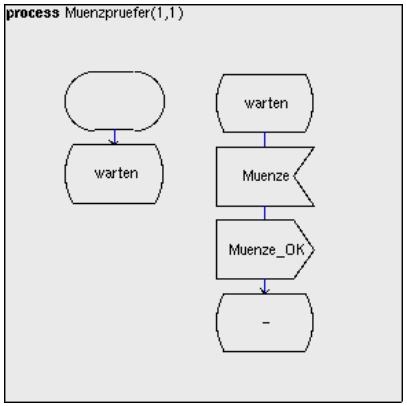
*Label* selektieren
Bitte selbständig eingeben
Doppelklick auf *betrag_anzeigen*
*Close*

Im *FrameWork*-Fenster eine der unteren 5 Zeilen selektieren, *File→Save*

## 4 Der Checker

Um zu testen, ob die Spezifikation syntaktisch (und grob semantisch) korrekt ist, soll nun der *Checker* angeworfen werden. (Syntaktische Fehler werden – wie euch vielleicht schon aufgefallen ist – im übrigen bereits bei der Eingabe bemängelt.) Der *Checker* findet die Fehler, die “zur Compile-Zeit” entdeckt werden können. “Laufzeit”-Fehler werden beim Verifizieren bzw. Validieren (s.u.) gefunden.



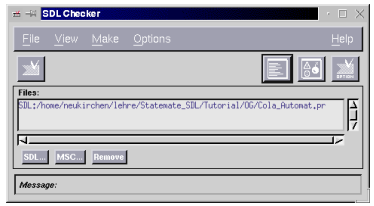



Im *FrameWork* im Menü *Tools* unter den Punkt *SDL & MSC Checker* aufrufen. Der *SDL Checker*-Dialog erscheint.

In diesem Fenster kann sofort das Icon *Check SDL and MSC description(s)* angeklickt werden.

Nach kurzer Zeit das *Errors*-Fenster mit der Anzahl der gefundenen Errors, Warnings und Informations. Falls alle drei Einträge ganz rechts *0* anzeigen, sind wir glücklich. Falls davor bereits Fehler angemahnt werden, kann per Doppelklick auf diese Zeilen der angezeigte Fehler im SDL-Diagramm lokalisiert werden. *Close* schließt das *Errors*-Fenster, das *SDL Checker*-Fenster muß mit *File→Exit* beendet werden.

*Tools→SDL & MSC Checker*




*Check SDL and MSC description(s)*
*Close*
*File→Exit*

## 5 Simulieren

Das erstellte SDL-Modell soll nun simuliert werden. Hierzu muß im *FrameWork* im Menü *Tools* unter den Punkt *SDL & MSC Simulator* aufgerufen werden. Das *SDL Simulation Builder*-Fenster, das dem *SDL Checker*-Fenster ähnelt, erscheint. Zusätzlich besitzt dieses Fenster jedoch den Button *Build a simulation executable*. Nach Klick auf diesen Button wird das SDL-Modell in eine ausführbare Datei umgewandelt.

(Prinzipiell könnte SDL natürlich auch interpretiert werden, aber gerade bei großen Systemen, die nicht nur interaktiv simuliert, sondern auch automatisch validiert (s.u.) werden sollen, wäre dies zu langsam!)

*Tools→SDL & MSC Simulator*

*Build a simulation executable*

Das bekannte *Errors*-Fenster sollte keine Fehler mehr melden und kann mit *Close* geschlossen werden.

*Close*

## 5.1 Simulator aufrufen

Der eigentliche Simulator wird mit dem Button *Start the simulation* aufgerufen. Daraufhin erscheint das Simulator-Fenster und der Simulator wartet auf Kommandos. (Evtl. Warnings bzgl. fehlender Startup-Dateien können ignoriert werden.)

*Start the simulation*

### 5.1.1 Anzeigeoptionen setzen

Da kein Mensch das komplette SDL-System im Kopf hat, wollen wir während des Simulierens wissen, wo wir uns gerade im SDL-Modell befinden: Dies geschieht im Simulator-Fenster über den Button *SDL tracking*. Daraufhin erscheint ein Fenster, in dem jeweils ein aktueller Ausschnitt des SDL-Systems angezeigt wird.


SDL tracking

(Dieses *SDL tracking* kann beendet werden, indem einfach das zugehörige *FSM*-Fenster geschlossen wird.)

Außerdem wollen wir immer die aktuell in den Eingabewarteschlangen vorliegenden Signale angezeigt haben: *Watch…* auswählen und im daraufhin erscheinden *Watch creation*-Fenster *Queues* auswählen. Das *Watch creation*-Fenster kann per *Close* geschlossen werden.

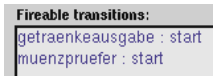

*Watch…*

*Queues*

*Close*

## 5.2 Das System ausführen

Im unteren Teil des Fensters zeigt der Simulator die ausführbaren Transitionen an. Da zunächst beide Starttransitionen ausgeführt werden sollen, den Eintrag *getraenkeausgabe: start* selektieren. Daraufhin werden die beiden Buttons für transitionsweise bzw. symbolweise Ausführung anwählbar.




*getraenkeausgabe: start*

### 5.2.1 Symbolweise ausführen

Zunächst wollen wir symbolweise durch das System steppen. Dies geschieht mit dem Button *step in* in der unteren Buttonleiste des Simulators, deshalb soll auch darauf geklickt werden.

*step in*

Im *FSM*-Fenster wird das Symbol angezeigt, das zur Ausführung ansteht, und im Textausgabefenster des Simulators wird ebenfalls dieses Symbol textuell umschrieben angezeigt. Der Beginn einer Transition wird jeweils durch *trans* und den dazugehörigen Transitionsnamen angezeigt.




Nun solange symbolweise das System ausführen, bis die aktuelle *Transition* beendet ist, d.h. der Zustand *warten* eingenommen wird. (Es wird immer das nächste *Symbol* angezeigt – nicht das aktuell ausgeführte!) Die *Step*-Buttons sind nun nicht mehr anwählbar.

*step in* bis zum Ende der *Transition*

Als nächstes durchlaufen wir auf dieselbe Weise die *Starttransition* des nächsten Prozesses; d.h. *muenzpruefer:start* selektieren und *step in* bis zum Ende der *Transition*.

*muenzpruefer:start* selektieren

*step in* bis zum Ende der *Transition*

Nun sind keine *Transitionen* mehr ausführbar, deshalb wird *Fireable transitions: deadlock* angezeigt.


### 5.2.2 Signale senden

Ein solcher “Deadlock” kann evtl. durch eine Eingabe, z.B. den Einwurf einer Münze, aufgehoben werden.

Dies geschieht durch *Output …* im Menüpunkt *Execute*. Das Fenster *Output* erscheint. Dort werden links oben zunächst die verschiedenen Signalquellen, wie das *Environment* oder einzelne *Prozesse* aufgeführt.

*Execute→Output …*

Da von außen ein *Signal* gesendet werden soll, muß *ENV* ausgewählt werden. Daraufhin kann rechts daneben das eigentliche *Signal* ausgewählt werden: *muenze*. Da in unserem Fall die *Signalroute* u.ä. eindeutig sind, kümmern wir uns nicht um die weiteren Einträge, sondern klicken auf *Apply*.

*ENV*

*muenze*

*Apply*

Da nun der nötige Stimulus vorliegt, kann das System wieder weiter ausgeführt werden. D.h. nun wird wieder eine ausführbare Transition angezeigt, die entsprechend selektiert werden kann.

### 5.2.3 Transitionsweise ausführen

Wer will, kann nun noch weiter symbolweise durch das System laufen und z.B. die Reaktion auf die Ausgabe des *Signal Getraenkanforderung* studieren. Schneller geht’s, indem transitionsweise durch das System gesteppt wird. Dafür ist das aber auch nicht so spannend, da das System im Zweifel nur in *warten*-Zuständen stehenbleibt und die nächste *Transition* noch nicht anzeigen kann, da diese von der nächsten Eingabe, die noch nicht feststeht, abhängt. In solch einer Situation bleibt die Anzeige auf dem jeweiligen *Nextstate*-Symbol stehen.

### 5.2.4 Die große Raubtiershow: Fütterung mit Signalen!

Da es auf Dauer ziemlich nervig ist, immer das *Output*-Fenster verwenden zu müssen, um anschließend im Simulator die zu feuernde *Transition* auswählen zu können, gibt es die Möglichkeit,

das SDL-System automatisch mit *Signalen* zu füttern, so daß im Simulator nur noch die jeweils zu feuernde *Transition* ausgewählt werden muß. Dies geschieht über das Fenster *Feed*, das im Menü *Edit* unter dem Punkt *Feed* ... zu finden ist.

*Edit→Feed ...*

In diesem Fenster ist zunächst – wie beim *Output*-Fenster – der *Kanal*, über den das System gefüttert werden soll, auszuwählen: *mechanik*. Daraufhin kann das eigentliche *Signal* ausgewählt werden: *muenze*. Da der Empfänger-*Prozess* in unserem Modell eindeutig ist, muß dieser nicht auch noch ausgewählt werden. Stattdessen reicht es, das ausgewählte *Signal* direkt per *Add*-Button zur *Feed List* hinzuzufügen.

*mechanik*
*muenze*

*Add*

Das gleiche passiert nun mit dem zweiten Signal: *elekronik* selektieren, daraufhin *getreankanforderung* auswählen und *Add*-Button drücken.

*elektronik*
*getreankanforderung*
*Add*

Mit *Close* kann das *Feed*-Fenster geschlossen werden.

*Close*

(Falls noch nicht geschehen, kann das Fenster *Output* per *Cancel* ebenfalls geschlossen werden, da es ja nun nicht mehr benötigt wird.)

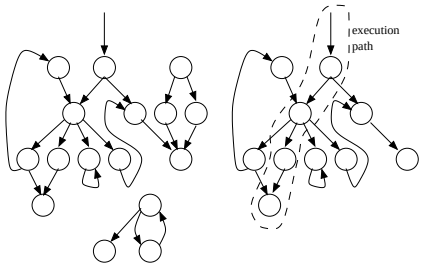
Von nun an sollten, die beiden *Signale* aus der *Feed List* automatisch beim Simulieren zur Auswahl stehen.

Kurz bevor Du die Lust am Simulator verlierst, solltest Du mit simulieren aufhören und den Simulator zum Verifizieren bzw. Validieren einsetzen.

# 6 Verifizieren bzw. Validieren

Das Verifizieren bzw. Validieren beruht auf dem der Simulation. Das Konzept der Verifizierung bzw. Validierung sieht vor, daß alle verschiedenen Zustände, die das SDL-System annehmen kann, herausgefunden werden, um festzustellen, ob es irgendwo Probleme geben könnte. (z.B. es liegt ein nicht behebbarer Deadlock vor oder es ist ein Zustand erreichbar, in dem ein Signal gesendet werden soll, das in dieser Situation von niemandem empfangen werden kann).

Fleissige können dies per Hand mit dem Simulator machen, indem alle Permutionen der jeweils möglichen Eingaben ausprobiert werden, ein wenig einfacher geht's mit der *Verify*-Option des Simulators. Diese ermittelt – ausgehend vom Startzustand – alle möglichen Folgezustände, indem ggf. die nötigen Eingaben gesendet werden, und setzt diese Suche rekursiv fort, um so den *Zustandsraum* bzw. den *Erreichbarkeitsgraphen* (aus dem sich alle möglichen *Ausführungspfade* ergeben) des Systems zu ermitteln.




(a) State diagram (b) Reachability graph



## 6.1 Vorbereitungen

Zunächst muß das System wieder in den Startzustand Zustand versetzt werden. Dies geschieht über den Button *init*.


*Init*

Der Simulator bietet die Möglichkeit, die Überdeckung des Systems anzugeben, d.h. wieviel Prozent der Symbole oder Transitionen bisher besucht wurden. Dies ist ganz praktisch, um "toten" Code zu finden, oder um beurteilen zu können, ob bei der Validierung auch alle Teile des Systems überprüft wurden. Dieser Wert soll ebenfalls zurückgesetzt werden: In der unteren Textzeile des Simulator `cover reset` eingeben.


`cover reset`

Außerdem muß – damit eine automatische Überprüfung stattfinden kann – die *Feed List*, wie in Abschnitt 5.2.4 angegeben, definiert sein.

*Feed List* definieren

## 6.2 Vollständige Verifizierung & Validierung

Zunächst soll der Simulator versuchen, den kompletten *Zustandsraum* des Automaten zu erkunden. Dies geschieht über den Menüpunkt *Verify* ... im Menü *Execute*.

*Execute→Verify ...*

Durch diesen Menüpunkt wird das Fenster *Verify Options* geöffnet. Es kann sofort der Button *Verify* gedrückt und anschließend bestätigt werden.

*Verify*, bestätigen

Darauf rattert der Simulator los und druckt alle paar Sekunden aus, wieviele Zustände er erreicht hat. Dieser Versuch der vollständigen Verfizierung & Validierung bricht nie ab. Daher muß der *Halt*-Button im Simulator-Fenster gedrückt werden, um den Validator abzubrechen.

Es erscheint eine Statistik im Textfeld. (Die genauen Zahlenwerte hängen davon ab, wann der Simulator abgebrochen wurde) Der Versuch der vollständigen Verfizierung bzw. Validierung mußte zwar abgebrochen werden, dennoch laut den drei *coverage*-Zeilen der Statistik wurde alle SDL-Transitionen, -Zustände und -Blöcke mindestens einmal besucht (100.00% coverage).

Transitions coverage rate : 100.00 (0 transitions not covered)
States coverage rate : 100.00 (0 states not covered)
Basic blocks coverage rate : 100.00 (0 basic blocks not covered)

Warum ist es nicht möglich, alle Zustände vollständig zu besuchen? Die Tatsache, daß dennoch 100% Systemüberdeckung vorliegt, deutet darauf hin, daß es schleifenartige Pfade gibt, die ziemlich bzw. unendlich lang sind. Solche Pfade sind möglich, da ja z.B. beliebig oft hintereinander Münzen in den Automaten geworfen werden, was dazu führt, daß die Variable *Betrag* hochgezählt wird und der Pfad der bisher besuchten Zustande, die sich jeweils in der Variablen *Betrag* unterscheiden, länger wird. Der *Zustandsraum* des Systems ist daher unendlich groß und kann nie vollständig verfiziert bzw. validiert werden!

## 6.3 Verifizieren & Validieren per Simulation

In solchen Fällen, in denen eine vollständige Validierung nicht möglich ist, bleiben verschiedene Möglichkeiten, zumindest Teile des Systems gezielt zu besuchen. Zum einen können Einschränkungen, z.B. durch komplexere, bedingte *Feed*-Kommandos, vorgenommen werden, zum anderen können bestimmte Pfade des Systems explizit überprüft werden.

Solche Pfade können durch Simulation gezielt durchlaufen werden.

Um auch noch zu einem späteren Zeitpunkt nachvollziehen zu können, welcher *Ausführungspfad* dabei durchlaufen wurde, kann der Button *Start MSC* gedrückt werden. Dadurch werden die im folgenden auftretenden Ereignisse “mitgeschnitten”.

*Start MSC*

Das Fenster *MSC Editor* erscheint und zeigt ein zunächst leeren *Message Sequence Chart* an.

Zunächst muß das System wieder in den Startzustand Zustand versetzt werden. Dies geschieht über den Button *init*.


*Init*

Es soll eine sinnvolle Benutzung des Cola-Automaten mitgeschnitten werden. D.h. es soll zunächst eine Münze eingeworfen werden, deren erfolgreiche Prüfung an die Getränkeausgabe mitgeteilt wird. Direkt im Anschluß soll ein Getränk angefordert werden,

Falls der zuletzt simulierte Pfad genau diesen Fall beinhaltet, kann der *Redo*-Button verwendet werden, um diese Schritte erneut “abzuspielen”.

Ansonsten bitte jetzt dieses Szenario mit dem Simulator durchspielen.

Szenario durchspielen

Falls man sich verklickt hat bzw. sich später noch entscheidet, doch einen anderen Pfad einzuschlagen, kann mit dem Button *Undo* auch zurück navigieren.

Nun soll der erzeugte Pfad, der in ObjectGEODE auch als Szenario bezeichnet wird, abgespeichert werden. Dies geschieht im Simulator-Hauptmenü: *File→Scenario→Save As...*. Als Dateiname `TraceLeadingToSuccess.scn` angeben.

*File→Scenario→Save As...*
`TraceLeadingToSuccess.scn`

## 6.4 Erfüllbarkeit von MSCs überprüfen

Wir haben nun mit Hilfe des Validators sichergestellt, daß es in unserem Modell einen Pfad gibt, der es erlaubt, nach Münzeinwurf ein Getränk am Automaten zu ziehen. Als letztes soll noch überprüft werden, ob es möglich ist, nur einmal zu bezahlen, aber zweimal hintereinander ein Getränk zu ziehen.



### 6.4.1 Szenarien in MSCs umwandeln

Damit der in Abschnitt 6.3 erzeugte Pfad visualisiert und editiert werden kann, soll die zugehörige Szenario-Datei in einen MSC umgewandelt werden. Mit *File→Scenario→Load* wird dieses Szenario *TraceLeadingToSuccess.scn* geladen.

*File→Scenario→Load*, *TraceLeadingToSuccess.scn*

Anschießend wird im Simulator-Hauptmenü: *File→Generate MSC ...* aufgerufen. Ein *Generate MSC*-Dialog erscheint.

*File→Generate MSC ...*

Zunächst muß dort der Dateiname, unter dem der zu generierende MSC abgespeichert werden soll, angegeben werden. Hierzu wird in das Feld rechts neben dem Button *Filename ...* `TraceLeadingToSuccess.msc` eingegeben und der Button *Generate* gedrückt. Im Fenster *FrameWork* erscheint ein neuer Eintrag für diesen MSC. Der *Generate MSC*-Dialog kann mit *Close* geschlossen werden.

rechts neben *Filename ...* `TraceLeadingToSuccess.msc` eingeben
*Generate*
*Close*

### 6.4.2 MSCs editieren

Um nun überprüfen zu können, ob es möglich ist, nur einmal zu bezahlen, aber zweimal hintereinander ein Getränk zu ziehen, muß der soeben erzeugte MSC modifiziert werden.

Hierzu muß der Simulator geschlossen werden, da es sonst nicht möglich ist, die Diagramme zu verändern. Im Simulatorfenster *File→Exit* auswählen und bei der Nachfrage *No* anklicken, damit wir uns nicht aus Versehen die Standard-Simulationseinstellungen zerschiessen.

im Simulatorfenster: *File→Exit*
*No*

Nun kann im *FrameWork*-Fenster der neu hinzugekomme MSC editiert werden: Ein Doppelklick auf die Zeile mit dem MSC *TraceLeadingToSuccess* öffnet das *MSC*-Editorfenster.

Doppelklick auf Zeile *TraceLeadingToSuccess*

Hinter die letzten im MSC aufgeführten Nachrichten fügen wird noch eine zweite Getränkanforderung samt (fälschlicher) Getränkausgabe hinzu:

*message*

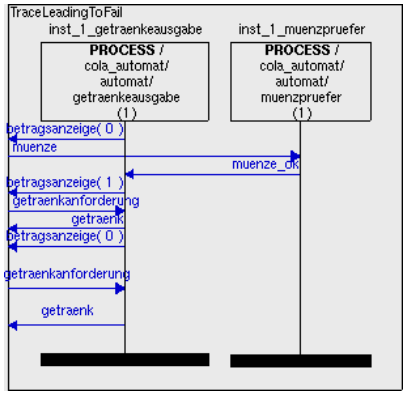
Das Icon *message* anklicken und vom *Environment* nach rechts zur Instanz *getraenkeausgabe* eine Nachricht für das *Signal* `getraenkanforderung` ziehen.

Klick auf *Environment*-Achse, , Klick auf *getraenkeausgabe*-Achse
`getraenkanforderung`

Ebenso von rechts nach links eine Nachricht *getraenk* anlegen.

Nachricht *getraenk* analog anlegen

Der resultierende MSC sollte wie folgt aussehen:




Dieser neue MSC muß nun per *File→Save As . . .* unter dem Namen `TraceLeadingToFail.msc` gespeichert werden.

*File→Save As . . .*
`TraceLeadingToFail.msc`

Im *FrameWork*-Fenster wird nun zwar der neue Dateinamen angezeigt, aber noch der alte MSC-Name. Deshalb dort die Zeile mit dem MSC selektieren und auf das Icon *Content* klicken. Dort kann dem MSC auch der neue Namen gegeben werden: `TraceLeadingToFail`.

Zeile mit MSC selektieren

*Content*
`TraceLeadingToFail`

Durch diese Aktion ist der ursprünglich MSC aus dem *FrameWork* herausgeflogen, so daß er per *File→Load* neu geladen werden muß.

*File→Load*

### 6.4.3 MSCs validieren

Für die beiden gespeicherten *MSCs* kann nun mit dem Validator überprüft werden, ob es im SDL-System *Ausführungspfade* gibt, mit denen sich deren Nachrichtenaustauschsequenz realisieren läßt.

Hierzu müssen die MSCs in das SDL-System einkompiliert werden. Dies geschieht – wie gewohnt – im Menü *Tools*unter den Punkt *SDL & MSC Simulator* und dem Button *Build a simulation executable*.

*Tools→SDL & MSC Simulator*

*Build a simulation executable*

Das bekannte *Errors*-Fenster sollte keine Fehler mehr melden und kann mit *Close* geschlossen werden.

*Close*

Der Simulator wird mit dem Button *Start the simulation* aufgerufen.

*Start the simulation*

Die *Feed List* muß wie zuvor konfiguriert sein.

*Feed List* konfigurieren

Dann kann der Menüpunkt *Verify . . .* im Menü *Execute*. aufgerufen werden. Das Fenster *Verify Options* erscheint. Rechts oben sollten unter *Observers* die beiden MSC-Namen aufgeführt sein.

*Execute→Verify . . .*


*Verify*, bestätigen

Der Button *Verify* kann gedrückt und anschließend bestätigt werden.

Darauf rattert wieder der Simulator los und kann per *Halt*-Button abgebrochen werden.

*Halt*

Interessant ist nun die Textausgabe bzgl. der beiden Observer (=MSCs): *traceleadingtofail* sollte *0 success* melden, was aussagt, daß kein Pfad gefunden werden konnte, der diesem MSC entspricht. *traceleadingtosuccess* sollte einige Male erfolgreich gewesen sein. D.h. es wurden Pfade gefunden die dem durch diesen MSC beschriebenen Szenario entsprechen.


## 7 Beenden

Per *Exit* im Menü *File* können die einzelnen Fenster des ObjectGEODE-Tools beendet werden. Evtl. Nachfragen zum Speichern können bestätigt werden, falls es noch ungespeicherte Änderungen gibt.

*File→Exit*