RoboLang oder so (mir ist noch immer nichts besseres eingefallen)

Roland Schweiger (Projektleiter)
Gerald Stieglbauer

17. März 2000

Zusammenfassung

Dieses Dokument beschreibt die Entwicklung einer graphischen Programmiersprache samt zugehöriger Programmierumgebung (IDE). Die Sprache ist speziell auf die Programmierung von Robotern abgestimmt. Es wird weiters eine Schnittstelle zu dem am Institut für Computerwissenschaften der Universität Salzburg entwickelten Robotersimulator DIMAS, sowie dem Robotor EMMA implementiert.

Verwendet wird die Programmiersprache JAVA und das Entwicklungstool <NetBeans/Forte for Java/JBuilder>. Als Output des graphischen Programms soll wiederum ein JAVA-Programm entstehen.

Inhaltsverzeichnis

Kapitel 1

Planungsphase

Hier werden folgende Punkte genauer behandelt:

- Welche Elemente werden in der graphischen Programmiersprache (GPS) verwendet?
- Was sind die genauen Funktionen und Eigenschaften dieser Elemente?
- Wie werden diese Elemente in die Programmierumgebung (IDE) eingebettet?
- Wie stellt sich der genaue Ablauf eines fertigen Programmes dar?
- Wie sehen die genauen Schnittstellen zu DIMAS und EMMA aus?

Dieses Kapitel versteht sich als abstrakte Festlegung von Grundkonzepten des Projektes und soll bereits eine innere Struktur der zu programmierenden Anwendungssoftware diskutieren. Wie groß das „Fenster“ zum Anwender der Software dann tatsächlich ist sowie die wichtigen Punkte wie Ergonomie, Benutzerfreundlichkeit, Zusatzprogramme, genaue Festlegung der Bedienelemente der IDE etc. wird im Kapitel 2 (Entwurfsphase) behandelt.

1.1 Elemente der GPS

Prizipiell lassen sich zwei Grundelemente unterscheiden:

- Units
- Connections (Verbindungen) zwischen den Units

1.1.1 Units

Eine Unit besteht also aus folgenden Komponenten (siehe Abb. 1.1):

- Objekttyp: Legt die allgemeine qualitative Eigenschaft der Unit fest
- Funktionstyp: Bestimmt die funktionelle Eigenschaft (Bezeichnung der Übertragungsfunktion) der Unit

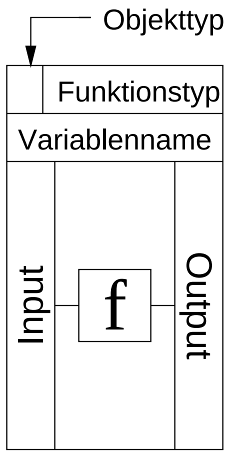


Abbildung 1.1: Schematische Darstellung einer Unit

- Variablenname einer Instanz einer speziellen Unit
- Eingangsparameter (Inputs)
- Ausgangsparameter (Outputs)
- Übertragungsfunktion
- ReihungsID (nicht in Abb. **??** dargestellt)

Objektyp: Unter allgemeine qualitative Eigenschaften versteht man die Art und Weise, wie eine Unit im Inneren aufgebaut ist. Prinzipiell wird zwischen folgenden Eigenschaften unterschieden:

- Roboter-Komponenten (Component-Units)
- JAVA-Programme (Java-Units)
- Fuzzy-Regler (Fuzzy-Units)
- Neuronale Netze (Net-Units)
- Super-Units
- Special-Units (Black-Boxes mit spezifisch festgelegter Übertragungsfunktion wie z.B. Kontroll-Units siehe 1.6)

Die ersten vier Punkte sind jeweils Spezialfälle des letzten Punktes. Units vom Typ Roboter-Komponenten sind Units, die aufgrund eines vorgegebenen Roboters in einer speziellen Roboterbibliotek zur Verfügung gestellt werden (wie z.B Roboterkamera, Motoren, etc).

Eine Super-Unit ist eine neue Unit, dessen innere Struktur aus mehreren verschalteten Units aufgebaut ist. Special-Units sind Units mit sehr speziellen, hier nicht weiter spezifizierten Eigenschaften, unter denen weitere Spezialfälle zusammengefaßt werden.

Funktionstyp: Unter einem Funktionstyp werden solche Units unter einem spezifischen Namen zusammengefaßt, deren Objekttypeigenschaften bereits vollständig definiert sind (z.B. ein fertiges JAVA-Programm, ein ausformulierter Fuzzy-Regler, ein fertig konstruiertes neuronales Netz etc.). In der objektorientierten Programmierung (OOP) würde das entsprechende Äquivalent dazu ein fertig programmiertes Objekt sein, von dem man nun mehrere Instanzen erzeugen kann.

Solche fertige Units können entweder vom Programmierer erstellt werden, oder können in Bibliotheken zur Verfügung gestellt werden und mit teilweise beschränktem Zugriff vom Programmierer modifiziert werden.

Eine Ausnahme stellen dabei die Component-Units dar. Von ihnen kann im gesamten Programm auf allen Ebenen nur eine Instanz erzeugt werden. Anschließend kann das spezifische Bauteil nicht mehr aus der Roboterbibliothek ausgewählt werden. Auch hat der Programmierer mit Ausnahme der Festlegung von Bauteil-Optionen (wie z.B. Auflösung einer Kamera etc.) keinen Zugriff auf die innere Struktur. Weiters ist der Name des Funktionstyps bereits festgelegt und kann nicht verändert werden.

Eine Ausnahme von dieser Ausnahme sind jedoch jene Component-Units, die nur Ausgänge, aber keine Eingänge besitzen. Bei dieser Art von Units ist es möglich mehrere Instanzen zu erzeugen, sonst werden sie aber wie normale Roboterbauteil-Units behandelt.

Variablenname: Ein Bezeichner der eine jeweilige Instanz einer Unit in der jeweils aktuellen Programmierebene eindeutig identifiziert.

Ausnahme hier wiederum die Component-Units: Deren Variablennamen sind bereit festgelegt und können nicht mehr verändert werden. Da i. a. jeweils nur eine Instanz in allen Programmierebenen erzeugt werden kann, ist auch der Variablenname in allen Programmierebenen eindeutig.

Eingangs/Ausgangsparameter, Übertragungsfunktion: Jede Unit besitzt eine variable Anzahl (darf auch null sein) von Eingangs- und Ausgangsparametern (oder Input- bzw. Outputpins). Die Ausgangparameter werden durch die Übertragungsfunktion anhand der Einganspararneter berechnet.

Jeder Eingangs- und jeder Ausgangsparameter hat einen bestimmten *Typ* (sh. 1.2). Ein einzelner Typ können sich auch aus mehreren Typen zusammensetzten.

Weiters besitzen die Eingangs- und Ausgangsparameter *Bezeichner*. Diese sind standartmäßig auf bis `i0` bis `ix` bzw. `o0` bis `ox` gesetzt, könner aber vom Programmierer unbenannt werden.

ReihungsID: Legt die bevorzugte Reihenfolge fest, in der die einzelnen Ausgangsparameter der Units anhand der Eingangsparameter berechnet werden (sh. 1.5.1).

1.1.2 Connections

Verbindungen (Connections) können von einzelnen Ausgangsparameter zu Eingangsparametern *gleichen* Typs gezogen werden. Dabei handelt sich um gerichtete Verbindungen, durch die Daten an weitere Units weitergeleitet werden.

Auf einer solchen Connection befindet sich ein einelementiger Datenspeicher der genau einen Datenwert vom Typ der Connections, der durch die vor- und nachgeschalteten Units bestimmt wird, aufnehmen kann.

Ändert sich der Datentyp eines Parameters einer Unit, so wird die Verbindung als ungültig markiert (aber nicht gelöscht), solange bis wieder eine Typkonsistenz gegeben ist.

Weiters besitzen Connections auch einen im jeweiligen Frame eindeutigen Bezeichner. Standartmäßig stimmt dieser mit dem Bezeichner des korrespodierenden Output-Pins überein. Handelt es sich um eine Connection die an einem Input-Feld angeschlossen wurde, so wird der Bezeichner des Pins des Input-Feldes dafür herangezogen. Der Programmierer kann diesen Bezeichner jedoch jederzeit neu benennen, solange dieser unter allen im Frame gezogenen Connections eindeutig bleibt.

1.2 Datentypen:

Jedem Eingangs- und Ausgangsparameter einer Unit muß ein Datentyp zugewiesen werden. Ein Datentyp kann sein:

- Atomarer Datentyp
- Zusammengesetzter Datentyp (Struktur)
- Ein- oder mehrdimensionale Arrays (Datenfeld)
- Datentypen vom Typ `intern`

Eine Gleichheit zwischen zwei Datentypen ist nur dann vorhanden, wenn es sich tatsächlich um eine identische Datentypdefinition handelt. Eine Umwandlung von atomaren Datentypen muß explizit durch eine zwischengeschaltet Unit erfolgen. Auch besteht bei zusammengesetzten Datentypen keine Gleichheit, wenn diese intern gleich deklariert wurden, jedoch nach außen verschiedene Namen aufweisen.

Eine auf allen Programmierebenen gleiche Datentypbibliothek soll die Datentypen verwalten und kann vom Programmierer erweitert werden. Atomare und vom System vorgegebene Datentypen können von dieser Datenbank weder gelöscht noch modifiziert werden.

1.2.1 Atomare Datentypen:

Es wird zwischen folgenden atomaren Datentypen unterschieden:

- `byte` (8 Bit)
- `short` (16 Bit)
- `int` (32 Bit)
- `long` (64 Bit)
- `float` (32 Bit)
- `double` (64 Bit)

- `char` (16 Bit)
- `boolean` (1 Bit)

1.2.2 Zusammengesetzte Datentypen oder Strukturen

Es sollen auch sogenannte Strukturen festgelegt werden können. Unter einer Struktur wird ein neuer Datentyp zusammengefaßt, der wiederum aus gültigen Datentypen besteht. Ein gültiger Datentyp ist rekursiv definiert als entweder ein atomarer Datentyp oder eine weitere Struktur.

In Bibliotheken sollen wiederum solche zusammengesetzte Datentypen dem Programmierer zur Verfügung gestellt werden. Weiters soll der Programmierer eigene Datentypen entwerfen und in Bibliotheken verwalten können. Dafür soll ein eigener Datentypeditor dem Programmierer als Hilfe gegeben werden.

1.2.3 Ein- oder mehrdimensionale Arrays

Natürlich können auch Datenfelder verwendet werden. Wesentlich dabei ist, daß die Größe und Dimension eines Arrays von Anfang an festgelegt wird. Datentypgleichheit besteht nur dann, wenn zwei Arrays den gleichen Datentyp für jedes Arrayelement verwenden und dieselbe Größe und Dimension aufweisen.

1.2.4 Interne Datentypen

Darunter fallen Steuerleitungen die sich auf die interne Struktur der Programmiersprache beziehen. Denkbar in diesem Zusammenhand wären zum Beispiel Steuerleitungen, die die internen Parameter von Membershipfunctions von Fuzzy-Reglern aktiv während des Programmablaufes verändern. Weiters sollen solche Datentypen bei sogenannten Kontroll-Strukturen eine wesentliche Rolle spielen (siehe 1.6). Es ist noch zu überlegen, ob und wie diese Datentypen eventuell in die oben genannten Typen konzeptuell eingegliedert werden könnten.

1.3 Konstanten und Initialwerte

Konstante und Initialwerte sind bei der hier zu entwerfenden graphischen Programmiersprache eng verwandte Begriffe.

Manchmal wird es notwendig sein Units Initialwerte zu geben, um mit Berechnungen beginnen zu können (so z.B. bei rekursiven Verbindungen). Deshalb kann man jedem Eingangsparameter einer Unit einen Initialwert zuweisen. Dieser muß natürlich mit dem Datentyp übereinstimmen. Für komplexere Datentypen, wäre es hier wierderum sinnvoll eine Datenbank mit für den jeweiligen Datentyp gültige Initialwerte zur Verfügung zu stellen, die wiederum vom Programmierer erweitert werden könnte.

Der Initialwert wird, falls eine Connection am Eingangsparameter anliegt genau einmal eingelesen. Falls keine Connection anliegt so wird der Initialwert als Konstante verwendet, die bei jedem Berechnungsschritt anliegt.

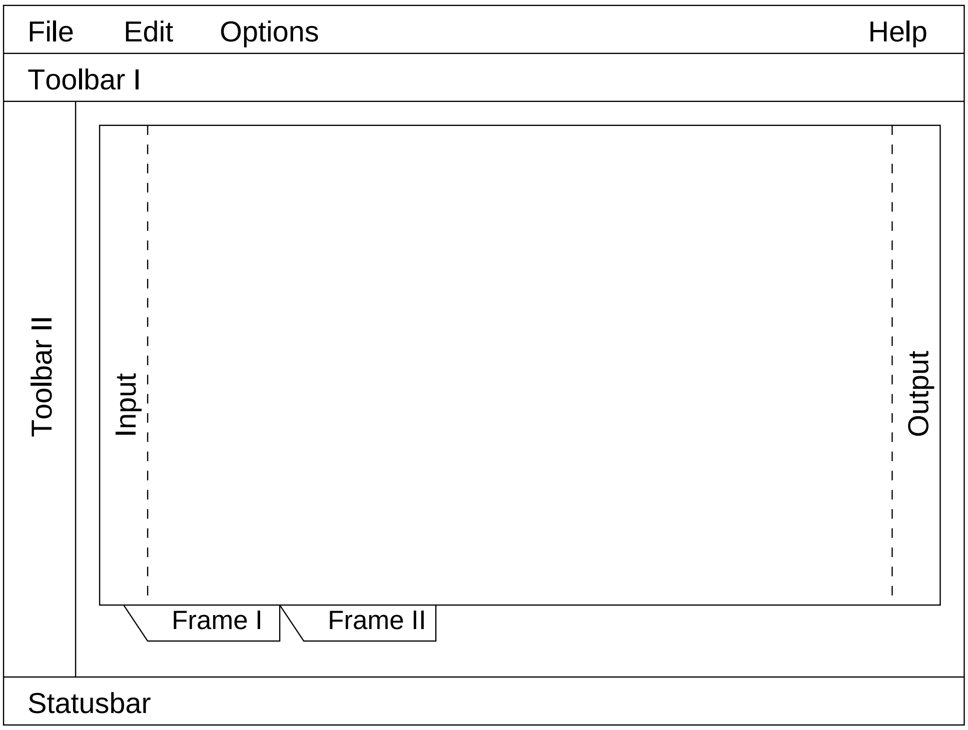


Abbildung 1.2: Schematischer Entwurf der IDE

1.4 Einbindung der Elemente in die IDE

Die IDE besteht prizipiell aus folgenden Elementen (siehe Abb. 1.2):

- Pull-Down Menüs: Stellt den gesamten Befehlssatz der IDE dar.
- Einer horizontalen Toolbar, in der die wichtigsten Elemente der Pull-Down Menüs vorkommen.
- Einer vertikalen Toolbar zur Auswahl und Manipulation von GPS-Elementen.
- Einer Statusleiste, zum Anzeigen von Hilfen und sonstigen Meldungen.
- Einer Framebereich, in der die Elemente der GPS plaziert werden können. Dieser besitzt weiters
  - eine Input-Leiste und
  - eine Output-Leiste.

1.4.1 Plazieren von Elementen

GPS-Elemente können nun in der vertikalen Toolbar aus einer Datenbank ausgewählt oder gänzlich neu erzeugt werden. Beim Auswählen aus einer Datenbank soll ein Libary-Manager aufgerufen werden, aus der der Programmierer bequem seine Elemente auswählen kann. Beim Erstellen von neuen Elementen muß der Programmieren zwischen den einzelnen Unit-Typen auswählen.

Danach kann das ausgewählte Element auf einer Frame plaziert (instanziert) werden. Durch Anklicken des Elementes können seine spezifischen Eigenschaften je nach Unit-Typ eingestellt oder programmiert werden.

Anschließend können Connections zwischen den Unit gezogen werden. Dies soll ebenfalls in der vertikalen Toolbar anwählbar sein.

Außerdem sollen in der vertikalen Toolbar noch diverse Buttons zur Manipulation der Anordnung der Elemente vorhanden sein (wie z.B. Löschen, Verschieben, Kopieren, etc.)

1.4.2 Programmierebenen

Bei der Verwendung von Super-Units muß zwischen verschiedenen Programmierebenen unterschieden werden:

- Mainframe
- Subframe

Mainframe: Das Mainframe stellt das Hauptprogramm dar. Bei kleineren Programmen wird man sich darauf beschränken können, nur eine Mainframe und keine Subframe zu verwenden.

Es ist aber auch erlaubt zur Verbesserung der allgemeinen Übersicht mehrere Mainframes zu verwenden (natürlich unter Überprüfung der Component-Unit-Abhängigkeiten wie unten in Subframe beschrieben). Es handelt sich dann dabei um zwei völlig getrennte Pfade, die in weiterer Folge als unabhängige Threads implementiert werden könnten, die aber auch keine direkte Möglichkeit zur Kommunikation besitzen.

Das Hauptprogramm kann diverse Aufrufparameter benötigen. Dies wird durch Ziehen von Connections von im Mainframe plazierten Units auf die linke Input-Leiste (sh. 1.3) des Frames ermöglicht. Weiters können beim Beenden des Programmes Rückgabewerte an den aufrufenden Prozeß übergeben werden. Dies geschieht analog durch ziehen von Connections an die rechte Output-Leiste.

Subframe: Will man nun zur übersichtlicheren Strukturierung eines Programms mehrere, in einem funktionellen Zusammenhang stehende Units zu einer einzigen Unit (Super-Unit) zusammenfassen, so ist die Verwendung von Subframes notwendig. Dieses unterscheidet sich hauptsächlich dadurch von einem Mainframe, da es nicht als eigenständiges Programm gestartet werden kann.

Ein Subframe definiert also eine neue Super-Unit und kann daher als Unit in einer User-Libary verwaltet werden. Somit kann dieses erzeugte Element in der Mainframe als eigenständige Unit eingesetzt werden. Weiters kann in einem Subframe auch auf andere bereits definierte Super-Units zugegriffen werden.

Die Eingangs- und Ausgangsparameter des neuen Unit-Blocks werden durch Ziehen von Connections im Subframe-Fenster nach der Input- und Outputleiste definiert. Somit werden dort auch die Datentypen festgelegt.

Einschränkungen sind bei der Verwendung von Component-Units zu beachten. Prinzipiell ist allgemein zu unterscheiden:

- Roboterspezifische Super-Units

- Roboterunabhängige Super-Units

Roboterspezifische Super-Units können nur ausgewählt werden, wenn alle in den Subunits vorkommenden Roboterbauteil-Units mit dem aktuell in Verwendung befindlichen Robotor übereinstimmen und diese auch noch nicht in irgendeiner, dem aktuellen Projekt zugehöriger Unit verwendet werden, da i. a. jede Component-Unit im gesamten Projekt nur ein mal verwendet werden darf.

Eine Ausnahme stellen, wie oben beschrieben jene Component-Units dar, die nur Ausgänge, aber keinerlei Eingänge besitzen. Diese können beliebig oft instanziert werden. Es gilt jedoch die Einschränkung, daß bei einer Veränderung von Bauteiloptionen einer solchen Unit sich diese Optionsänderung bei allen Instanzen gleichzeitig auswirkt.

Roboterunbahängige Super-Units enthalten keine Roboterbauteil-Units und können daher beliebig oft instanziert werden.

Es ist zu überlegen, wie oben genannte Abhängigkeiten für den Programmierer übersichtlich dargestellt werden könnten.

Projekt: Zu einem Projekt gehören alle neu programmierten Frames (sowohl Mainframe als auch Subframe). Es können weiters Units und Datentypen aus anderen Projekten importiert werden, sofern sie in den entsprechenden Datenbanken eingetragen wurden; auf deren Source kann jedoch aus leicht einzusehenden Gründen nicht zugegriffen werden. Weiters sind bei der Verwendung von Super-Units die oben genannten Einschränkungen zu beachten.

Weiters kann ein Projekt unter Berücksichtigung aller verwendetet Roboterbauteil-Units ein anderes Projekt als Super-Unit einbinden.

1.5 Runtime-Strategien:

Die Punkt umfaßt einen Lösungsansatz zu folgende Fragen:

- Wie wird nun das fertige Projekt zur Laufzeit abgearbeitet?
- In welcher Reihenfolge werden die Unit zur Berechnung der Output-Daten anhand der Input-Daten und der Übertragungsfunktion herangezogen?
- Wie funktioniert die Datenübergabe von einer Unit zur nächsten?
- Wie werden Super-Units abgearbeitet?

1.5.1 Vergabe von ReihungsIDs

Die ReihungsID soll eine Richtlinie angeben, in welcher Reihenfolge die einzelnen Units ausgeführt werden sollen.

Zu erst wenden wir uns dem einfachsten Fall zu: Es existiert nur ein Frame - das Mainframe. Es werden weiters keine Super-Units verwendet. Außerdem gibt es keine Verzweigungen, Rückkopplungen oder sonstige Schleifen. Ein Beispielprogramm ist in Abb. 1.3 dargestellt.

In welcher Reihenfolge werden nun die Units abgearbeitet? Es sind folgende zwei Grundstrategien denkbar:

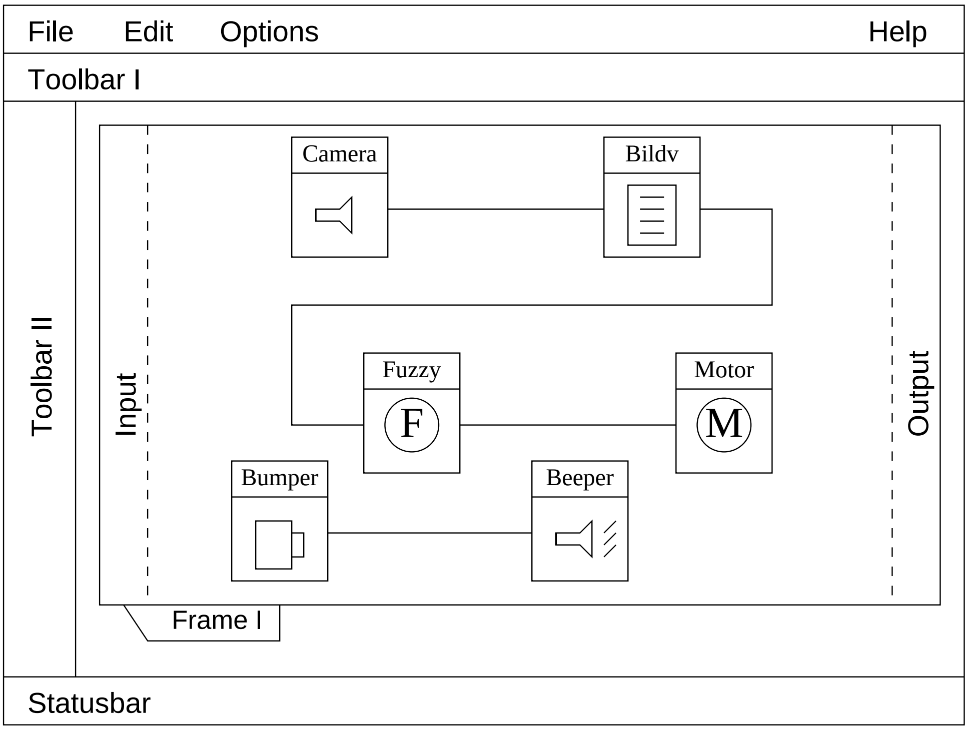


Abbildung 1.3: In diesem Beispielprogramm existieren zwei völlig voneinander unanbhängige Pfade. Der obere Pfad soll ein Bild mithilfe einer Kamera einlesen. Das Bild wird dann in einer Bildverarbeitung umgewandelt und das Ergebnis wird einem Fuzzy-Regler mitgeteilt. Dieser steuert schließlich den Motor des Roboters an. Der zweite Pfad überträgt nur Bumperimpulse an einen Lautsprecher.

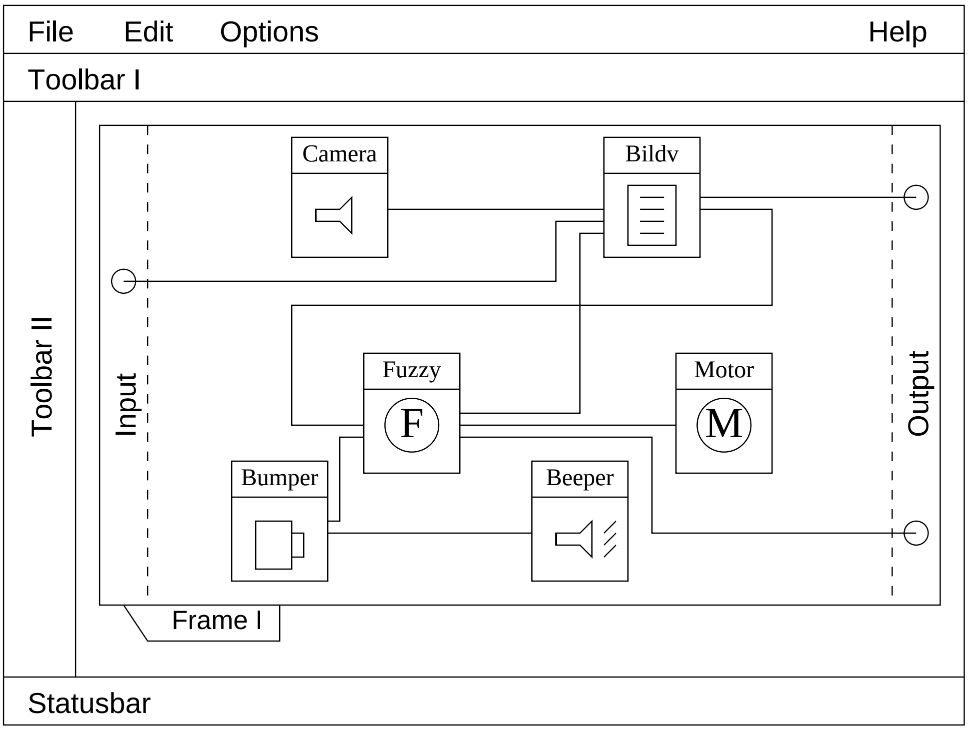


Abbildung 1.4: Erweiterung des Beispiels in der Abb. 1.3. Es soll hier weniger ein praktisches Beispiel als vielmehr die Möglichkeiten komplizierter Verknüpfungen (wie Rekursion, etc.) dargestellt werden.

- Automatische Vergabe
- Manuelle Vergabe

Für jedes Frame soll nun vom Programmierer entschieden werden welche Strategie angewendet werden soll. Bei der manuellen Vergabe soll ein Unitmanager für das jeweilige Frame den Programmierer dabei unterstützen. In diesem Unitmanager besteht also die Möglichkeit die zu bevorzugende Abarbeitungsreihenfolge zu definieren.

Die manuelle Vergabe soll allerdings nur in Spezialfällen eingesetzt werden (z.B. für eine Zeitoptimierung). Bei der automatischen Vergabe wird nun nach Algorithmus 1 vorgegangen, der auch kompliziertere Fälle wie in Abb. 1.4 behandelt.

Welche Strategie in den speziellen Unterteilungen des Algorithmus 1 jeweils verwendet wird, ist vom Programmierer abhängig und kann durch Optionen in einzelnen Units bzw. der Frames festgelegt werden. Es wird von Anwendungsfall zu Anwendungsfall verschieden sein, welche Strategie die bessere ist.

Algorithm 1

1. Absolute Priorität haben jene Units die entweder keine Eingänge besitzen, deren Eingangsparameter mit dem Inputfeld des Frames verbunden sind oder deren Eingangsparameter ausnahmslos mit Initialwerten oder Konstanten belegt sind. Sie sind der *Beginn* eines Pfades.

2. Anschließend wird Schrittweise entlang des durch die gerichteten Verbindungen vorgegebenen Pfades entlanggegangen und somit die Numerierung fortgesetzt. Sind mehr als ein Pfad vorhanden, so wird ein beliebiger Pfad ausgewählt, um einen Schritt fortzuschreiten. Danach wird der nächste Pfad ausgewählt und genauso verfahren.

3. Ist das *Ende* eines Pfades erreicht, so wird dieser vorübergehend stillgelegt, bis alle übrigen Pfade abgearbeitet sind. Danach sind alle Units numeriert worden. Das Ende eines Pfades ist erreicht, wenn

   - es keine weiteren Connections mehr gibt
   - das Outputfeld des Frames erreicht worden ist
   - das Innere Gebiet einer Kontrollstruktur (sh. später) betreten wird
   - oder die nachfolgende Unit schon durch einen anderen Pfad erschlossen und numeriert worden ist.

4. Teilt sich ein Pfad in zwei oder mehrere Pfade so gibt es wiederum zwei Strategien:

   (a) Der geteilte Pfad gliedert sich in zwei oder mehrere Subpfade, deren virtuelles Erscheinungsbild wieder einen einzigen Pfad ergibt. Bei der Numerierung kommt dieser virtuelle Pfad nach oben genannter Strategie zum Zug; intern kommt allerdings erst der erste, beim nächsten Numerierungschritt der zweite usw. dran.

   (b) Die durch die Teilung neu entstandenen Pfade werden völlig gleichwertig in die bereits vorhandenen Pfade eingefügt und kommen wie gehabt bei der Numerierung an die Reihung.

5. Bei rekursiven Verbindungen entscheidet über eine weitere Numerierung dieser rekursiven Verbindung die Tatsache ob die an dieser rekursiven Verbindung anliegende Unit schon berücksichtigt worden ist oder nicht. Eine rekursive Verbindung ist also nur ein Spezialfall eines endenden Pfades.

6. Tritt eine Super-Unit auf so sind wieder zwei Strategien möglich:

   (a) Die Unit wird in seine Bestandteile aufgelößt und in das Mainframe virtuell hineinkopiert. Die so neu entstandenen Units werden so wie bereits in der Mainframe vorhandene Units behandelt.

   (b) Die Unit bleibt weiterhin als Einheit geschlossen und wird bei der Abarbeitung vollständig berechnet, sodaß am Ausgang alle Werte anliegen. Intern wird wieder wie beginnend mit Punkt 1) verfahren.

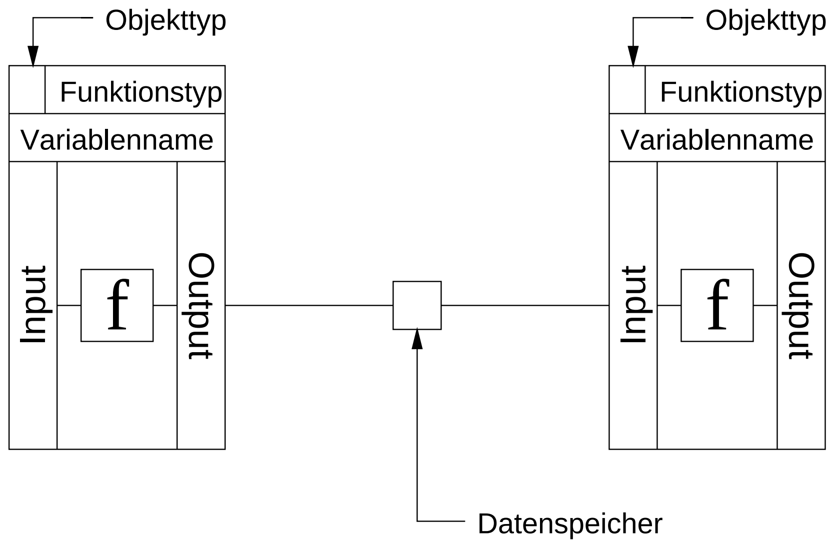


Abbildung 1.5: Schematische Darstellung zu Datenübertragung zwischen zwei Units: Das kleine Viereck soll den einelementigen Datenspeicher darstellen, indem die Daten einer Unit an eine andere weitergegeben werden. Natürlich entstehen bei mehreren Connections mehrere solche Datenspeicher.

1.5.2 Parameterübergabe zwischen Units

Zwischen jeden Units, die durch eine Connection miteinander kommunizieren, befindet sich auf jeder Verbindung ein Datenspeicher der genau ein Element des zugehörigen Datentyps aufnehmen kann. Wie man sich das graphisch vorstellen kann ist in Abb.1.5 dargestellt.

Wie zur Laufzeit dann die einzelnen Units ausgewählt werden und die Parameterübergabe genau funktioniert, wird durch Algorithmus 2 bestimmt.

1.6 Kontrollstrukturen:

Die Programmiersprache soll auch die sprachüblichen Kontrollstrukturen aufweisen. Dazu gehören:

- if - Kontrollstruktur (mit nicht ausschließenden Nachfolgepfaden)
- if - Kontrollblock (mit ausschließenden Nachfolgepfaden)
- while - Kontrollblock
- do-while - Kontrollblock
- for - Kontrollblock

Bei den Kontrollstrukturen ist also weiters zu unterscheiden:

- Einelementige Kontrollstruktur (hier einfach Kontrollstruktur)
- Mehrelementige Kontrollstruktur (Kontrollblock)

Algorithm 2

1. Eine Scheduler zur Verwaltung der einzelnen Unit-Aufrufe wird gestartet. Dieser wählt nun die einzelnen Units zur Berechung ihrer Ausgangsparameter anhand der vergebenen ReihungsID aus. Ist das letzte Element erreicht, so wird danach wieder von vorne begonnen.

2. Bevor eine Berechnung einer gewählten Unit gestartet wird werden zwei Bedingungen überprüft (Abarbeitungs- oder Ausführbedingung):

   (a) An allen Eingängen liegt ein Wert an. Ein Wert liegt an, wenn die Datenspeicher zwischem der vorgeschalteten Units und der ausgewählten Unit belegt ist, ein Initialwert oder eine Konstante angegeben wurde, oder der Unit durch das Inputfeld des Mainframes ein Argument übergeben worden ist. Handelt es sich um eine Unit deren Eingang mit dem Inputfeld des Subframes verbunden ist, so liegt dann ein Wert an, wenn die im übergeordneten Frame eingesetzte Super-Unit die gleichen Bedingungen wie eine normale Unit vorfindet.

   Sind keine Eingänge vorhanden, so ist die Bedingung immer erfüllt.

   (b) Die zwischen der gewählten Unit und der nachgeschalteten Units befindlichen Datenspeicher ist leer bzw. einer oder mehere der Datenspeicher ist nicht vorhanden (d. h. es besteht keine Connection an einem vorhandenen Output-Pin). Sonderfälle werden wie in (a) behandelt.

   Sind keine Ausgänge vorhanden, so ist die Bedingung immer erfüllt.

3. Sind die beiden Bedingungen von 2. erfüllt, so wird die Berechnung der Ausgänge anhand der Eingänge und der Übertragungsfunktion berechnet. Andernfalls wird bei 2. mit der nächsten vom Scheduler ausgewählten Unit fortgesetzt.

4. Alle Datenspeicher zwischen der vorgeschalteten und der ausgewählten Unit werden gelöscht. Weiters werden die Ergebnisse der Berechungen in die zugehörigen Datenspeicher der Ausgänge geschrieben. Es ist erlaubt, daß nicht alle oder auch keine Werte in die nachfolgenden Datenspeicher geschrieben werden (z.B. Component-Unit Kamera liefert nur alle 200ms ein Bild).

5. Nun wird bei 2. mit einer vom Scheduler neu gewählten Unit fortgesetzt.

Die einelementige Kontrollstruktur - besteht im wesentlichen aus einer Special-Unit (hier *Kontroll*-Unit). Diese Unit weist im Gegensatz zu herkömmlichen Units etwas flexiblere Abarbeitungsvorschriften sowie eine ganz spezifische interne Übertragungsfunktion auf. Die Eigenschaften dieser Übertragungsfunktion soll im wesentlichen über Optionsfenster der jeweiligen Unit spezifiziert werden können (d.h. kein Java-Programm o. ä.).

Auch ist es bei solchen Unit möglich über *Steuerleitungen* (Connections vom Typ `intern`) gewisse Input-Pins (Eingangsparameter) einer Unit ein- bzw. auszuschalten, was sich vor allem bei der Abarbeitungsbedingung auswirkt, da ausgeschaltete Pins nicht in dies Bedingung miteinfließen. Auch kann eine solche Steuerleitung durch einen spezifischen Wert die Berechung der Übertragungsfunktion unterdrücken. Wichtig ist, daß immer erst der Wert der Steuerleitung eingelesen und ausgewertet wird, bevor die anderen Input-Pins gelesen werden. Wenn weiters die Übertragungsfunktion nicht ausgeführt wird, so wird auch wie in Algorithmus 2 speziefiziert, der Datenspeicher zwischen der aktuellen und der vorgeschalteten Unit nicht gelöscht, d. h. auch Werte der Steuerleitungen liegen weiter an der jeweiligen Unit an.

Es kann weiters vorkommen, daß solche Units auch interne Variablenwerte statisch halten; d. h. daß Werte zwischen zwei erfolgreichen Abarbeitungen der Übertragungsfunktion gespeichert werden.

Die mehrelementige Kontrollstruktur - oder *Kontrollblock* besteht aus mehreren durch Steuerleitungen miteinander verknüpften Kontroll-Units. Sie bilden somit einen *Block*. Ein Block ist graphisch gesehen ein begrenztes Areal in dessen Inneren zusätzliche Gesetze bei der Plazierung von Units und Verbinden von Units durch Connections herrschen können. Tritt eine beliebige Connection direkt in einen solchen Block ein, so bedeutet dies das Ende eines Pfades.

Optisch bilden im folgenden die Steuerleitungen diese optischen Begrenzungen, dies ist aber willkürlich und kann in der entgültigen Benutzer-IDE anders gelöst werden. Die Steuerleitungen selbst existieren immer und es kann immer nur der gesamte Block, nie aber eine einzelne Units daraus plaziert werden.

1.6.1 Die if - Kontrollstruktur

Diese einelementige Kontrollstruktur könnte prinzipiell auch als normale Unit realisiert sein. Um eine gewisse konzeptuelle Einheit zu waren, wird sie jedoch zu den Kontrollstrukturen hinzugerechnet.

Bei der if-Kontrollstruktur handelt es sich um die primitivste denkbare bedingte Verzweigung in der GPS. Sie besitzt keine Ein- oder Ausgänge für Steuerleitungen.

Die Unit kann beliebig viele normale Eingänge besitzen. In dem Optionenfenster der Unit können diese Inputs durch ihere Bezeichner logisch miteinander verknüpft werden. Bei der logischen Verknüpfung werden die standartmäßigen Verknüpfungsoperatorern wie `AND, OR, EXOR, NOT` etc. zur Verfügung gestellt.

Es können in diesem Optionenfenster beliebig viele Bedingungen formuliert werden. Aus der Anzahl der Bedingungen ergibt sich dann die Anzahl der Output-Pins plus einem zusätzlichen else-Pin der Kontroll-Unit. Standartmäßig ist der Typ der Ausgangsparameter `boolean`. Dies kann jedoch vom Programmierer für jeden Pin einzeln verändert werden.

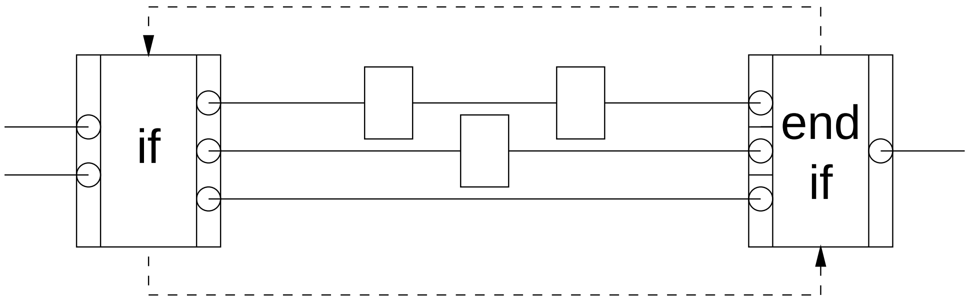


Abbildung 1.6: Einfacher Anwendungsfall eines if - Kontrollblockes.

Ist nun bei der Abarbeitung die if-Kontroll-Unit an der Reihe, so wird wie in Algorithmus 2 definiert die Abarbeitungsbedingung überprüft. Ist diese erfüllt, so werden anhand der Eingangswerte die formulierten logischen Werte der bereits definierten Bedinungen berechnet. Ist für die Ausgansparameter kein bestimmter Wert und Typ spezifiziert, so wird im Falle des Werts `true` einer bestimmten Bedingung, der korrespondierende Ausgangspin mit dem Wert `true` beschrieben. Ist keine der Bedingungen „wahr", so wird der else-Pin mit einem Wert beschrieben.

Dieser geschriebene Wert kann aber vom Programmierer noch beliebig näher (durch eine Konstante oder einen arithmetischen Ausdruck) für jede Bedingung definiert werden.

Wesentlich bei der einfachen if-Kontrollstruktur ist, daß alle Bedingungen auf ihren Wahrheitswert überprüft werden und somit beliebig viele nachgeschaltete Pfade parallel weiterverfolgt werden können. Es handelt sich also um eine nicht ausschließende oder-Verknüpfung der Bedingungen.

1.6.2 Der if - Kontrollblock

Der *if-Kontrollblock* (siehe Abb. 1.6 und Abb. 1.7) besteht nun aus zwei durch Steuerleitungen verbundene Kontroll-Units:

- Die if-Kontrollunit und
- die end-if-Kontrollunit.

Die if-Kontroll-Unit: Diese Unit stimmt im wesentlichen mir der in 1.6.1 definierten Kontroll-Unit überein. Es bestehen aber auch einige gravierenden Unterschiede:

1. Die Unit besitzt eine eingehende und eine ausgehende Steuerleitung. Der interne Typ der eingehenden Connection ist wie ein Verbindung vom Typ `boolean` realisiert; die ausgehende Steuerleitung ist intern vom Typ `integer`. Liegt ein Wert an der eingehenden Steuerleitung an, so wird wie in 1.6.1 verfahren, sonst ist klarerweise die Ausführbedinung nicht erfüllt.

2. Bei der Berechung des Wahrheitswertes der im Optionenfenster der Unit formulierten Bedingungen wird solange beginnend mit der ersten Bedingung solange fortgefahren, bis eine wahre Bedingung (falls keine „wahr"

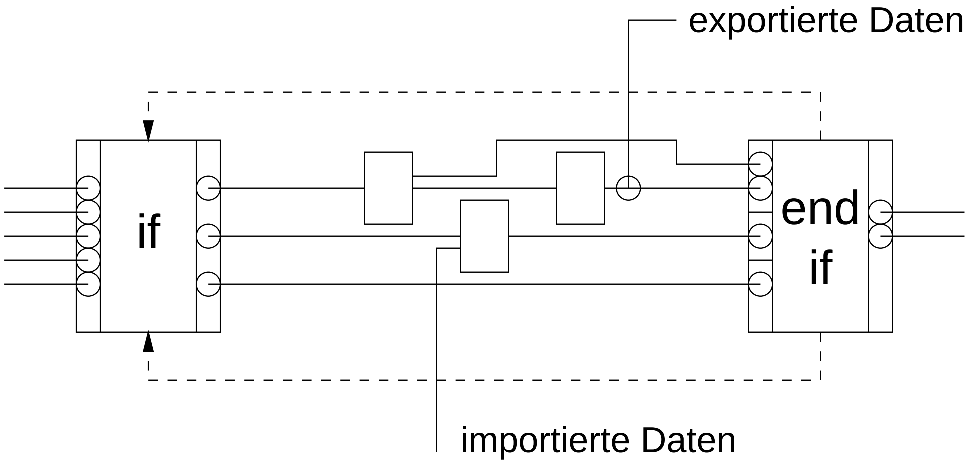


Abbildung 1.7: Komplizierterer, durch angegebene Definition des if-Kontrollblocks mit maximaler Funktion versehener Anwendungsfall.

ist so wird wieder der else-Pin mit einem Wert beschrieben) erreicht ist. Nun wird abgebrochen und nur der zu dieser Bedingung korrespondierende Ausgang gesetzt. Die restlichen Ausgänge bleiben unberücksichtigt. Die Bedingungen sind also im Gegensatz zu 1.6.1 durch ein ausschließendes „oder“ miteinander verknüpft.

3. Nach der Überprüfung der Bedingungen wird an der ausgehenden Steuerleitung ein mit der wahren `x`-ten Bedingung korrespondierender Integer-Wert `x` geschrieben.

Die end-if-Kontroll-Unit: Diese Unit ist etwas komplizierter aufgebaut, um möglichst flexible Programmiermöglichkeiten zu erreichen. Sie besitzt beliebig, wie immer von dem Programmierer konfigurierbare Eingangsparameter. Diese Parameter können weiters in Gruppen zusammengefaßt werden. Jeder dieser Gruppe wird ein Integer-Wert zugeordnet. Optisch sind diese Gruppen in Abb.1.6 und Abb. 1.7 durch Strichunterteilungen getrennt.

Es kann immer nur eine Gruppe zur Bewertung der Ausführbedingung herangezogen werden. Welche Gruppe ausgewählt wird, wird durch die eingehende Steuerleitung entschieden.

Die Anzahl und Typ der Ausgänge sind ebenfalls variabel und können vom Programmierer eingestellt werden. Im Optionenfenster kann dann (eventuell graphisch) festgelegt werden, welcher Eingang mit welchem Ausgang korrelliert. Hier soll auch noch eine Standartvorgebung realisiert werden, um dem unbedarften Benutzer Arbeit abzunehmen (genaueres dazu folgt in der nächsten Version dieses Dokuments).

Ist die Ausführbedingung erfüllt so werden zuerst die korrellierenden Ausgangspins mit den Werten der ausgewählten Eingangsgruppe beschrieben. Anschließend wird auf der ausgehenden Steuerleitung (die sich wie eine Connection vom Typ `boolean`) verhält der Wert `true` geschrieben.

Allgemeines: Im Inneren des Strukturblocks, dem Strukturgebiet, ergeben sich nun von der Anzahl x der Ausgänge der if-Unit x neue Pfade. Da immer nur ein Pfad gleichzeitig abgearbeitet wird, ist es nicht sinnvoll, die Pfade untereinander mit Connections zu verbinden. Das gilt natürlich nicht für die neuen Pfade, die entstehen, wenn einer der vorher genannten Pfade sich erst später wieder in neue Pfade spaltet.

Auch müssen alle Endigungen dieser unmittelbar nach der if-Unit entstehenden Pfade in einer Gruppe der end-if-Unit enden, um eine Blockierung bei der Ausführung auszuschließen. Es ist zu diskutieren, inwieweit man dem Programmierer daran hindern soll, solche Programmzustände zu formulieren.

Es ist weiters erlaubt, Daten außerhalb des Kontrollgebietes zu importieren oder zu exportieren (sh. Abb. 1.7).

Wie sich nun der gesamte if-Kontrollblock zur Ausführungszeit verhält ist in Algorithmus3 beschrieben.

Algorithm 3

Wird in der nächsten Version des Dokumentes formuliert .

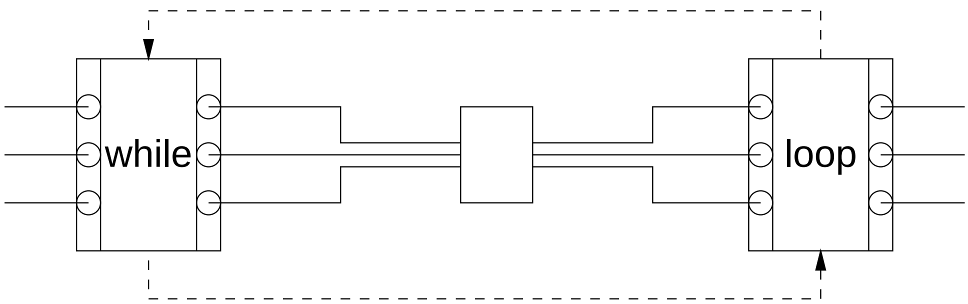
1.6.3 Der while - Kontrollblock (noch zu Überarbeiten; sh. nächste Version)

Der while - Kontrollblock besteht wie der if - Kontrollblock aus zwei Spezialunits (Black-Boxes):

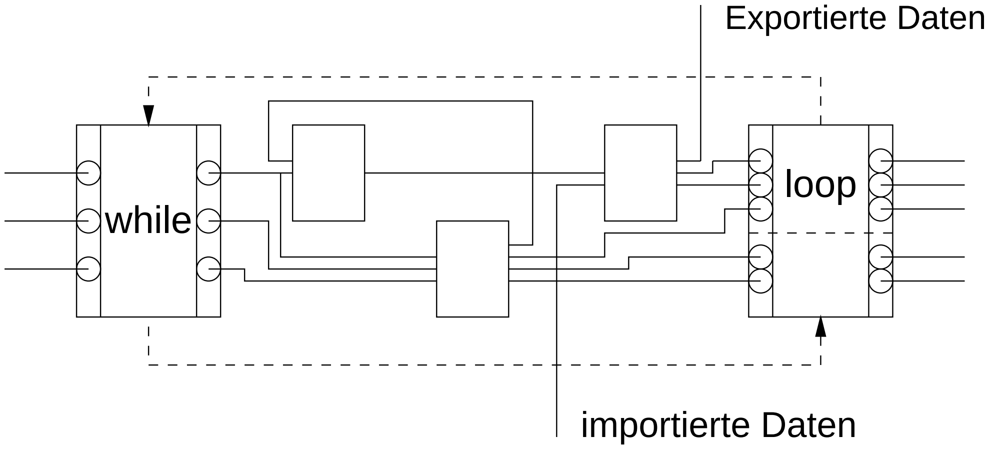
- Eine while-Unit und
- eine loop-Unit.

Schematische Darstellung:

- Einfacher Fall



- Komplizierterer Fall



Die while-Unit: - besitzt beliebig viele normale Eingangsleitungen, die der Programmierer ständig erweitern und reduzieren kann. Die Leitungen können aus beliebigen Typen bestehen.

In einem Optionenfenster der while-unit können diese Eingänge logisch zu einer Bedingung verknüpft werden. Hierbei sollen die Verknüpfungregeln genauso wie bei der if-Unit realisiert sein, mit der Ausnahme, daß nur eine Bedingung definiert werden kann.

Weiters ist die Anzahl und Typ der Output-Leitungen der while-Unit identisch mit den der Input-Leitungen.

Zudem besitzt die while-unit eine eingehende und eine ausgehende Steuerleitung vom Typ intern. Dieser ist eine zusammengesetzte Struktur, die zum einen aus einem Boolschen und zum anderen aus einem eindimensionalen Array besteht, dessen Größe identisch mit der Anzahl der Eingänge der while-Unit ist. Der Boolsche Wert besitzt den Initialwert "false".

Die ausgehende Steuerleitung ist vom Typ her genauso, wie die eingehende Steuerleitung definiert.

Liegt nun an allen Eingängen ein Wert an UND ist der Boolsche Wert der eingehenden Steuerleitung mit einem Wert "false" belegt, so werden diese Werte in einem internen Datenspeicher der while-Unit geschrieben.

Ist dieser Boolsche Wert "true", so wird, falls Werte in dem Array der Steuerleitung anliegt, das Array auf den internen Datenspeicher geschrieben.

Dieser Boolsche Wert fungiert also zum steuern eines Demultiplexers. Man könnte zudem diese interne Steuerleitung als zwei normale Connections realisieren, mit der Ausnahme, daß die Boolsche Connections bei der standartmäßigen Überprüfung der Ausführbedingung bei Units eine Sonderbehandlung erfährt. Die Übergabe der Werte dieser Sonderleitungen funktioniert aber wie im oben genannten Konzept der Parameterübergabe zwischen zwei Units. Da nun dieser Sondereingang speziell behandelt wird, ist die Ausführbedingung dieser Unit nie erfüllt falls kein Boolscher Wert anliegt und es werden keine Werte, weder von den normalen Eingängen, noch vom Array der Sonderleitungen eingelesen.

Ist andererseits der Boolsche Wert zwar belegt, die Eingänge sind oder das Array der Steuerleitung ist jedoch nicht belegt, so ist die Ausführbedingung ebenfalls nicht erfüllt. Da aber die einelementigen Zwischenspeicher der Connections erst dann gelöscht wird, wenn die nachgeschaltete Unit ausgeführt wurde, so bleibt aber der Boolsche Wert der Steuerleitung weiterhin anliegend.

Nun wird die in der while-Unit formulierte Bedingung anhand der zwischengespeicherten internen Werte auf ihren Wahrheitsgehalt überprüft. Der Boolsche Teil der ausgehenden Steuerleitung wird daraufhin mit dem Ergebnis dieser Berechnungen belegt.

Ist der Wert "true", so werden die Ausgänge der while-Unit mit den Werten des internen Speichers belegt. Ist der Wert jedoch "false", so wird am Ausgang nichts geschrieben, das Array der Steuerleitung aber mit den internen Speicherwerten belegt.

Die loop-Unit: Die Anzahl und der Typ der Eingänge der loop-Unit ist im einfachsten Fall mit Anzahl und Typ der Eingänge in der while-Unit identisch und auch davon direkt abhängig. Das heißt, daß eine Veränderung der Eingänge der while-Unit eine Veränderung der Eingänge der loop-Unit bewirkt. Zusätzlich kann der Programmierer die Anzahl der Eingänge noch vergrößern aber nicht verringern. Die zusätzlichen Eingänge sollen auch optisch von den anderen Eingängen getrennt sein, auf die der Programmierer keinerlei modifizierenden Einfluß haben soll.

Anzahl und Typ der Ausgänge setzt sich weiters, ähnlich wie bei der while-Unit aus den Eingängen der loop-Unit zusammen.

Zudem besitzt die loop-Unit wiederum eine eingehende und eine ausgehende Steuerleitung. Der Typ ist mit den der Steuerleitungen der while-Unit identisch. Die Enitäten der Steuerleitungen besitzen keinen Initialisierungswert.

Liegt nun am Boolschen Teil der eingehenden Steuerleitung der Wert "true" an, so werden alle fixen Eingänge (also alle jene Eingänge der Existenz von der while-Unit abhängen) auf das Array der ausgehenden Steuerleitung geschrieben. Weiters wird der Boolsche Teil auf "true" gesetzt. Die eventuell vorhandenen Werte der zusätzlichen Eingänge werden eingelesen und statisch innerhalb der loop-Unit zwischengespeichert.

Liegt am Boolschen Teil der eingehenden Steuerleitung jedoch der Wert "false" an, so wird das Array der eingehenden Steuerleitung auf die korrespondierenden Ausgänge geschrieben. Weiters werden die eventuell zwischengespeicherten Werte der zusätzlichen Eingänge der loop-Unit auf die restlichen Ausgänge übertragen. Abschließend wird der Boolsche Wert der ausgehenden Steuerleitung mit "false" belegt.

Allgemeines: Prinzipiell gibt es im Gegensatz zum if-Kontrollblocks im Innerern der Kontrollblocks keine verschaltungstechnischen Einschränkungen.

Wie nun der gesamte Block zur Ausführungszeit abgearbeitet wird, ist in Algorithmus 4 beschrieben

Algorithm 4

Nähere Ausführungen folgen in der nächsten Version des Dokumentes.

1.6.4 Der do-while - Kontrollblock:

Folgt in der nächsten Version.

1.6.5 Der for - Kontrollblock:

Folgt in der nächsten Version.