RoboLang oder so

Stieglbauer Gerald

10. 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.

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 der Programmiersprache. Die wichtigen Punkte wie Ergonomie, Benutzerfreundlichkeit, Zusatzprogramme, genaue Festlegung der Bedienelemente der IDE etc. werden in Kapitel 2 (Entwurfsphase) behandelt.

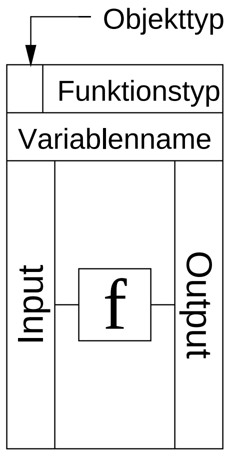
1.1 Elemente der GPS

Prizipiell lassen sich zwei Grundelemente unterscheiden:

- Units
- Verbindungen (Connections) zwischen den Units

1.1.1 Units

Schematisch sieht eine Unit folgendermaßen aus:



Eine Unit besteht also aus folgenden Komponenten:

- Objekttyp: Legt die allgemeine qualitative Eigenschaft fest
- Funktionstyp: Bestimmt die funktionelle Eigenschaft
- Variablenname einer Instanz einer speziellen Unit
- Eingangsparameter
- Ausgangsparameter
- Übertragungsfunktion
- ReihungsID

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

- Roboterbauteile
- JAVA-Programme
- Fuzzy-Regler
- Neuronale Netze
- Super-Units
- Black-Box mit spezifisch festgelegter Übertragungsfunktion (z.B. Kontrollblöcke)

Die ersten vier Punkte sind jeweils Spezialfälle des letzten Punktes. Roboterbauteile sind Units, die aufgrund eines vorgegebenen Roboter in einer speziellen Roboterbibliotek zur Verfügung gestellt werden.

Eine Super-Unit ist eine neuer Unit-Block, dessen innere Struktur aus mehreren Unit-Blöcken aufgebaut ist. Black-Boxes 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 Roboterbauteile 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 Roboterbauteile, 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 Roboterbauteile-Units: Deren Variablennamen sind bereit festgelegt und können nicht mehr verändert werden. Da 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 von Eingangs- und Ausgangsparametern. Die Ausgangparameter werden durch die Übertragungsfunktion anhand der Eingansparameter berechnet.

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

Weiters besitzten 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 Reihenfolge fest, in der die einzelnen Ausgangsparameter der Units anhand der Eingangsparameter berechnet werden (sh. später).

1.1.2 Connections

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

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.

1.2 Datentypen:

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

- Atomarer Datentyp
- Zusammengesetzter Datentyp
- Ein- oder mehrdimensionale Arrays
- Interne Datentypen

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 Datentype zusammengefaßt, der wiederum aus gültigen Datentypen besteht. Ein gültiger Datentyp ist 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 als 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. 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

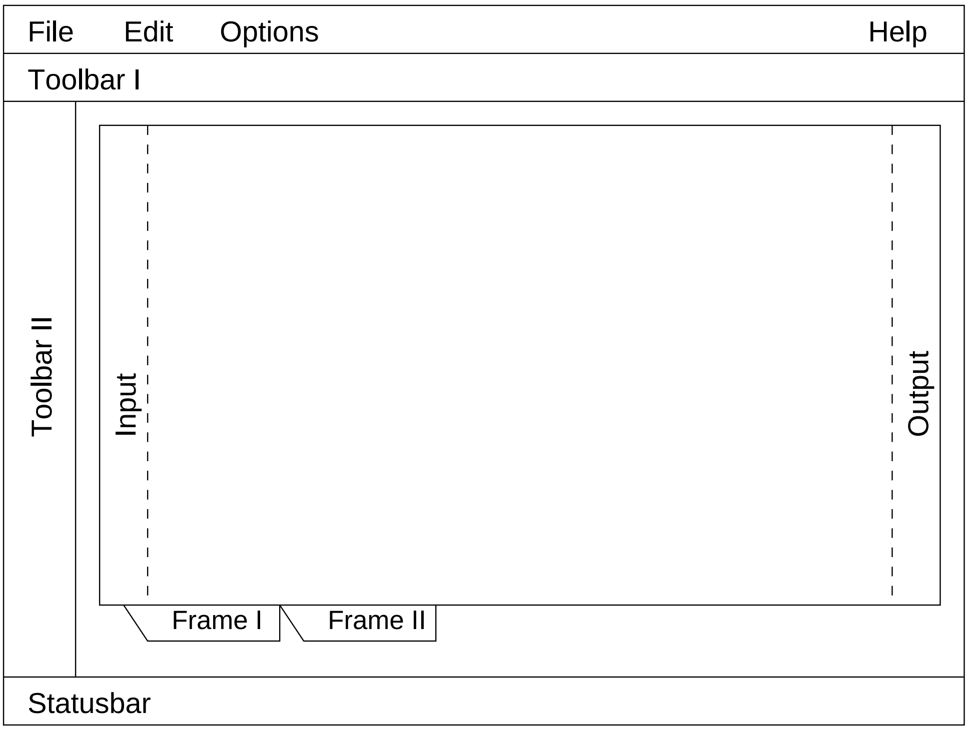
Konstante und Initialwerte sind bei der hier zu entwerfenden graphischen Programmiersprache eng verwandte Begriffe.

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 gültigenm 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.

1.4 Einbindung der Elemente in die IDE

Schematischer Entwurf der IDE:



Die IDE besteht prizipiell aus folgenden Elementen:

- 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.

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 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.

Das Hauptprogramm kann diverse Aufrufparameter benötigen. Dies wird durch ziehen von Connections von im Mainframe plazierten Units auf die linke Input-Leiste 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 übersichtlicher 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 Roboterbauteil-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 jedes Roboterbauteil-Unit im gesamten Projekt nur ein mal verwendet werden darf.

Eine Ausnahme stellen, wie oben beschrieben jene Roboterbauteil-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 beachen.

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

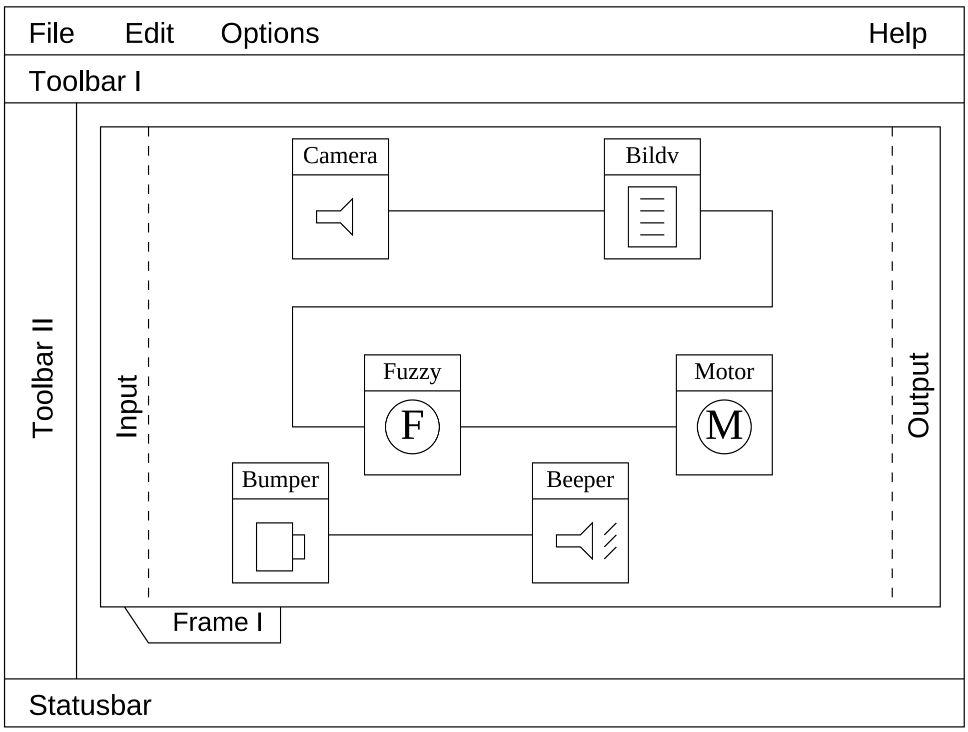
1.5 Runtime-Strategien:

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

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 könnte so aussehen:



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

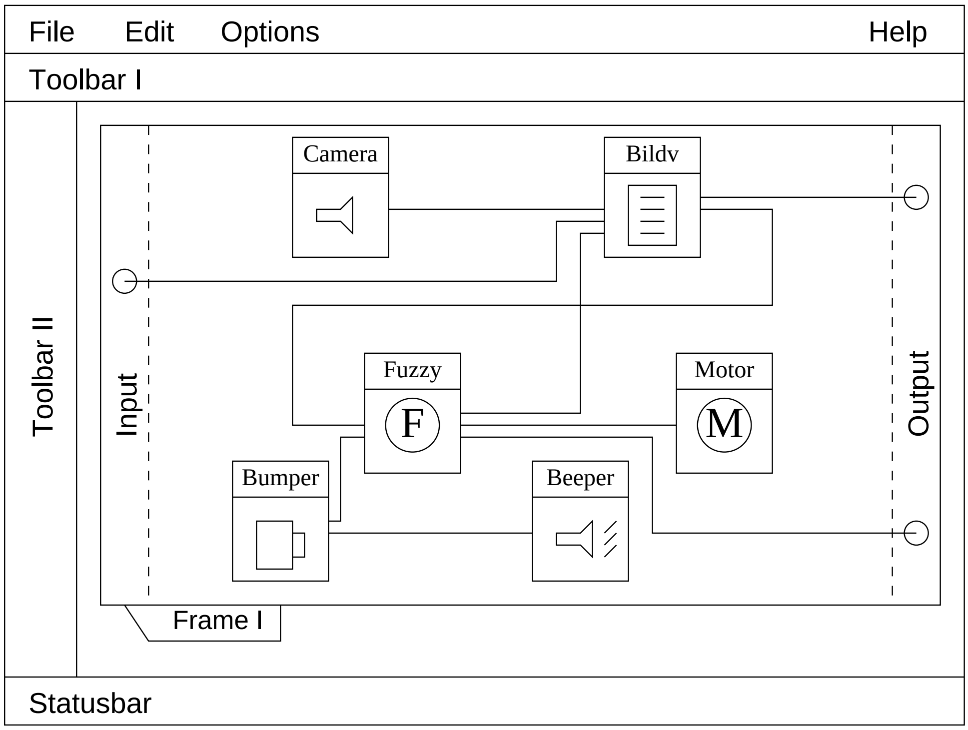
- Automatische Vergabe
- Manuelle Vergabe

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 gibt es nun wiederum mehrere Strategien:

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.
2. Anschließend wird Schrittweise entlang des durch die gerichteten Verbindungen vorgegebenen Weges 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, oder die nachfolgende Unit schon durch einen anderen Pfad erschlossen und numeriert worden ist.

   Komplizierter wird es bei folgenden Sonderfällen:



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:

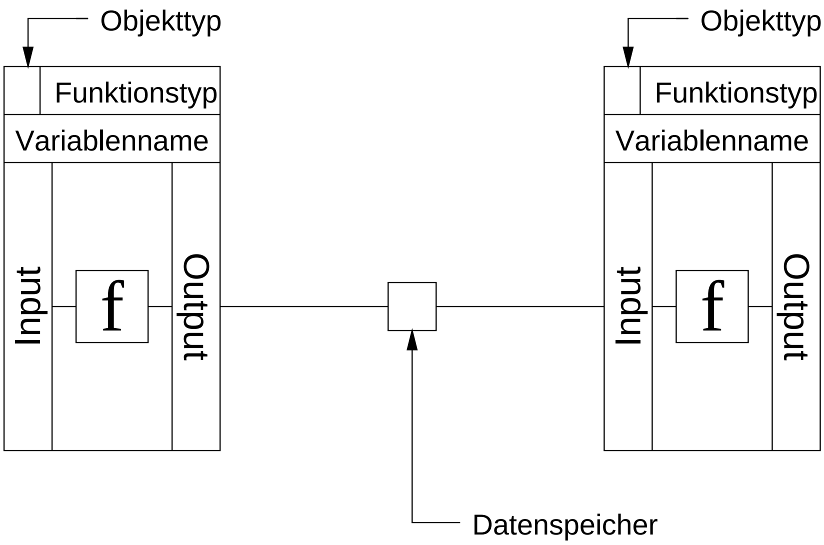
   (a) Die Unit wird in seine Bestandteile aufgelößt und in das Mainframe virtuell hineinkopiert. Die so neu entstandenen Units werden so wie bereits in der Mainframe vorhandene Units behandelt.

   (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.

Welche Strategie in den Spezialfällen jeweils verwendet wird, ist vom Programmierer abhängig und kann durch Optionen der einzelnen Units festgelegt werden. Es wird von Anwendungsfall zu Anwendungsfall verschieden sein, welche Strategie die bessere ist.

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. Schematisch kann man sich das so vorstellen:



Zur Laufzeit geschieht dann folgendes:

1. Eine Scheduler zur Verwaltung der einzelnen Unit-Aufrufe wird gestartet.
2. Dieser wählt nun die einzelnen Units zur Berechung ihrer Ausgangsparameter anhand der vergebenen ReihungsID aus. Ist das letzte Element erreicht, so wird wieder von vorne begonnen.
3. Bevor eine Berechnung einer gewählten Unit gestartet wird werden zwei Bedingungen überprüft:

   (a) An allen Eingängen liegt ein Wert an. Ein Wert liegt an, wenn der Datenspeicher zwischem der vorgeschalteten Unit 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) Der zwischen der gewählten Unit und der nachgeschalteten Unit befindliche Datenspeicher ist leer. Sonderfälle werden wie in (a) behandelt.

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

4. Sind die beiden Bedingungen von 3. erfüllt, so wird die Berechnung der Ausgänge anhand der Eingänge und der Übertragungsfunktion berechnet. Andernfalls wird bei 2. fortgesetzt.

5. Alle Datenspeicher zwischen der vorgeschalteten und der ausgewählten Unit werden gelöscht. Weiters werden die Ergebnisse der Berechungen in die zugehörigen Datenspeicher der Ausgänge geschrieben.

6. Nun wird bei 2. fortgesetzt.

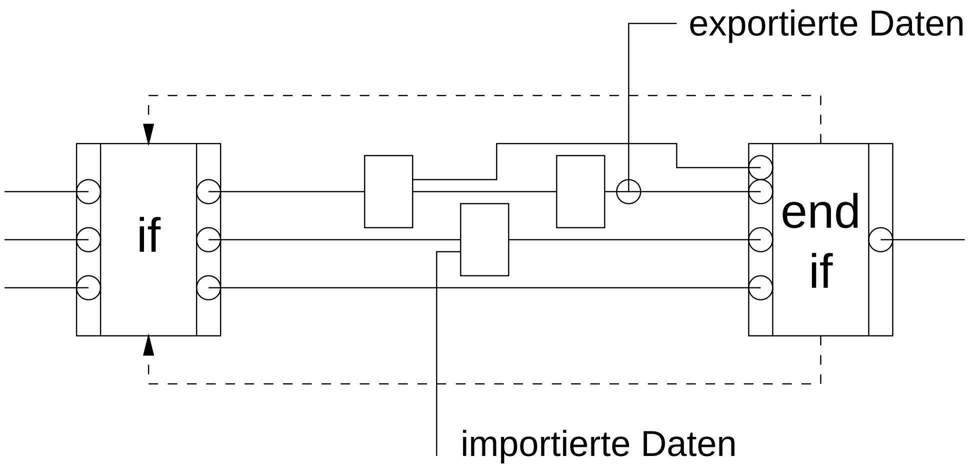
1.6 Kontrollstrukturen:

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

- if - Kontrollblock
- while - Kontrollblock
- for - Kontrollblock
- do-while - Kontrollblock

1.6.1 Der if - Kontrollblock

Schematisch kann man sich den IF - KONTROLLBLOCK folgendermaßen vorstellen:



Jeder hier dargestellte Block ist im Grunde nichts anderes als eine eigene Unit mit spezifischen Eigenschaften. Bei der Abarbeitung dieses Kontrollblockes wird folgendermaßen vorgegangen:

1. Es können beliebig viele Eingangsleitungen an den IF - KONTROLLBLOCK angeschlossen werden. Liegen an allen Eingänge Werte an so können verschiedene Bedinungungen überprüft werden. Diese können im Optionenfenster des if-Unit festgelegt werden. Dort werden die Variablenname der Eingänge mit logischen Operatoren verknüpft. Die Anzahl der Ausgänge der if-Unit hängt von der Anzahl der dort formulierten logischen Verknüpfungen ab.

2. Nun werden die einzelnen Bedingungen der Reihe auf ihren Wahrheitswert überprüft. Ist eine Bedingung wahr so wird ein ebenfalls im Optionenfenster festgelegter Wert eines bestimmten Typs am korrespondierenden Ausgangs der if-Unit ausgegeben (standartmäßig der Wert `true` vom Typ `boolean`) und der entsprechende Pfad des IF - KONTROLLBLOCK ausgeführt. Ist keine der Optionen wahr, so wird der letzte Ausgang der if-Unit mit einem Wert belegt.

3. Wie oben beschrieben wird eine Unit nur dann ausgeführt, falls alle Eingänge einen Wert aufweisen. Da die if-Unit rekurente Leitungen von der end-if-Unit als Eingang besitzt, müssen auch diese einen Wert haben, um die if-Unit zu aktivieren. Daher sind die Initialwerte dieser Verbindung standartmäßig auf `true`. Nach dem Bearbeiten der if-Unit befindet sich also kein Wert in dem Datenspeicher und die Unit ist solange blockiert, bis ein Pfad vollständig durchlaufen ist. Dies ist notwendig, um zu verhindern, daß zwei Pfade im IF - KONTROLLBLOCK gleichzeitig ausgeführt werden, was zu falschen Ergebnissen führen kann.

   Ist ein Pfad vollständig durchlaufen, so liegt am Ausgang der end-if-Unit der Wert `true` an und die if-Unit steht wieder zur Verfügung. Spaltet sich innerhalb eines IF - KONTROLLBLOCK ein Pfad, so gelten die Teile des Pfades als ein einziger Pfad.

   Importierte Daten (sh. Abb.) sind erlaubt, während Verknüpfungen zwischen Pfaden innerhalb des IF - KONTROLLBLOCK verboten sind. Auch sind exportierte Daten erlaubt; sie sind jedoch im allgemeinen mit Vorsicht zu genießen, da nicht immer so leicht abschätzbar ist, welcher Pfad ausgeführt wird.

4. Jeder Pfad im IF - KONTROLLBLOCK hat also eine beliebiege Anzahl von Endigungen in der end-if-Unit (da Spaltung erlaubt ist). Die Anzahl der Ausgänge der end-if-Unit entspricht daher der maximalen Anzahl von Endigungen Pfaden im IF - KONTROLLBLOCK. Die durchgeschalteten Werte der end-if-Unit entsprechen den Werten der Endigungen des aktiven Pfades. Ausgänge die aufgrund einer niedrigeren Anzahl von Endigungen eines Pfades nicht benötigt werden, werden einfach nicht beschrieben (d.h. der nachgeschaltete Dateinspeicher bleibt leer.)

Es soll im späteren Verlauf überlegt werden, wie man diese if - Konstruktion so standartisiert, daß Benutzerfreundlichkeit und volle Funktionsfähigkeit gleichzeitig gegeben sind (etwa durch Verdecken von Spezialfunktionen und Propagierung von Standartsituationen). Auch ist zu prüfen, wieweit die oben angeführten Bedingungen möglichst viele Programmiersituationen abdecken.