

Funktionsnetze in Verteilten Systemen

Heinrich P. Godbersen

TFH Berlin

Interner TFH-Bericht

TFH Berlin, FB Informatik, Luxemburger Str. 10, D 13353 Berlin; e-mal: godberse@tfh-berlin.de

0 Zusammenfassung:

Mit zunehmender Verbreitung von Rechnernetzen wird die Entwicklung verteilter Systeme immer wichtiger. Eine Software-Produktionsumgebung auf der Basis von Funktionsnetzen (FN) bietet einen Ansatz, die Komplexität dieser Aufgabenstellung zu reduzieren und dabei gleichzeitig den Erstellungsprozeß zu flexibilisieren und effizienter zu gestalten.

Ziel ist es, eine Umgebung zu schaffen, in der verteilte Anwendungen auf einer graphischen Oberfläche spezifiziert, synthetisiert und unmittelbar ausgeführt werden können.

Dieses Bericht faßt das Forschungsthema Funktionsnetze in vielfältiger weise zusammen. Ausgangspunkt ist eine Einführung und Skizze der "alten" Einsatzgebiete Modellierung von Entscheidungsunterstützungssystemen (Simulation). und Programmierung im Kleinen. Daran schließt sich eine Übersicht zum Stand der Technik bezüglich der Architektur von Verteilten Systemen an. Dies beinhaltet auch die Skizzierung von Kommerziellen Ansätzen zur Graphischen Programmierung. Schließlich wird ein Konzept zum Einsatz von FN zur Programmierung im Großen innerhalb Verteilter Umgebungen entwickelt. Dazu gehört die Vorstellung einer Workbench. Ein ausführliches Literaturverzeichnis schließt die Arbeit ab.

Die Ziele dieser Arbeit sind:

1. Diskussionsgrundlage (work in progress)
2. Dokumentation der Fortschreibung der Forschungsaktivitäten
3. Einarbeitung für Diplomanden

Dieser interne TFH-Bericht ist auch im Internet publiziert (<http://www.tfh-berlin.de/~godberse>).

Inhalt (Gesamtdokument): Funktionsnetze in Verteilten Systemen

1 EINFÜHRUNG

1. [Klassische Funktionsnetze](#)
2. [Geschichte](#)

2 MODELLE ZUR ENTSCHEIDUNGSUNTERSTÜTZUNG

2.1 [Ziele bei der Modellbildung](#)

2.2 [Deskriptive Modelle](#)

2.3 [Operationale Modelle](#)

2.4 [Benutzerinteraktion](#)

3 PROGRAMMIEREN IM KLEINEN

3.1 [Aufgabenstellung](#)

3.2 [Lösungsansätze](#)

3.3 [Schnittstellen](#)

4 STAND DER TECHNIK

1. [Visual Programming](#)

2. [Middleware](#)

5 [VERTEILTE FUNKTIONSNETZE](#)

5.1 [Anforderungen](#)

5.2 [Lösungen mit Funktionsnetzen](#)

5.3 [Realisierungskonzepte](#)

5.4 [Ausblick](#)

6 [DIE WORKBENCH VERSOS](#)

6.1 [Basis](#)

6.2 [Vorgehensmodell bei der Softwareerstellung](#)

6.3 [Die Workbench](#)

7 [LITERATUR](#)

7.1 [Direkter Bezug zu Funktionsnetzen](#)

7.2 [Hintergrundliteratur](#)

Copyright ©; Godbersen, 1996 Stand 29.09.96 0.htm

1 EINFÜHRUNG

lokaler Inhalt:

1.1 [Klassische Funktionsnetze](#)

1.1.1 [Beispiel: Fibonacci Zahlen](#)

1.1.2 [Zusammenfassung der Attribute](#)

1.1.3 [Vergrößerung und Verfeinerung](#)

1.2 [Geschichte](#)

[zurück zum Gesamt-Inhaltsverzeichnis](#)

Es wird eine Einführung in grundlegende Konzepte gegeben. Weiterhin wird kurz über die Geschichte der Forschungsaktivität berichtet.

1.1 Klassische Funktionsnetze

Aufbauend auf die Instanz/Kanal-Interpretation im Rahmen von Petrinetzen wurden für Funktionsnetze (FN) folgende neue Konzepte eingeführt (s. Abb.1.1):

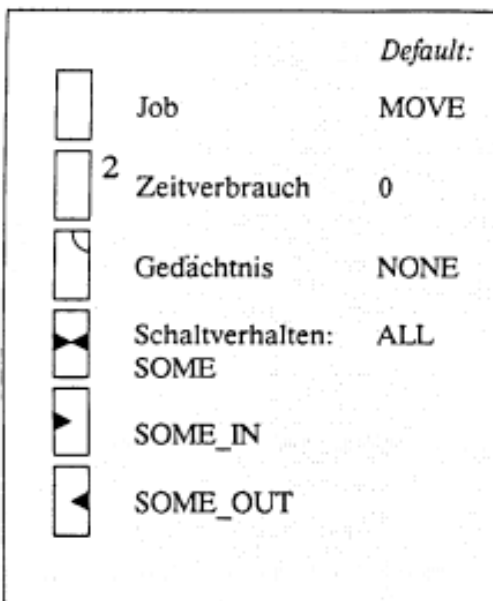
1. Um die Netzgröße insgesamt überschaubar zu halten, sehen wir diverse **Attribute** bei den Knoten und Systembeziehungen vor.
2. Um die Marken strukturieren zu können, führen wir eine **Nachrichten-Anschrift** ein (vgl. Attribut "**Initialisierung**")
3. Die Einbeziehung dynamischer Aspekte ermöglichen wir durch den Fluß der Nachrichten entlang der gerichteten Wege; **Operationen** auf den Nachrichten werden ausschließlich in den Instanzen durchgeführt (Attribut "**Job**").
4. Den Instanzen (aktiven Funktionseinheiten) wird ein **Zeitverbrauch** zugeordnet, um auch die Zeit konzeptionell in das Modell einzubeziehen (z.B für Leistungsvorhersagen, Attribut "**Time**").
5. Um die Dynamik in einem System adäquat modellieren zu können, fahren wir neben dem 'UND'-Schaltverhalten ein **Partielles Schalten** ein, das 'ODER'-Verknüpfungen realisiert.

- Die Partielle Schaltregel erlaubt insbesondere die adäquate Modellierung von Selektion und Verteilung. Attribute "**ALL, SOME, SOME_IN, SOME_OUT**"
6. Um unterschiedliche Formen von Systembeziehungen realitätsnah zu modellieren, unterscheiden wir zwischen **Nachrichten-** und **Steuerfluß** Attribute: **Input, Output, ..** sowie verschiedenen **Zugriffsformen** auf Kanäle. Attribute **FIFO, LIFO**
 7. Das Konzept der **Vergrößerung** und **Verfeinerung** erlaubt die Beherrschung der Komplexität durch Abstraktion, ohne dabei notwendigerweise inhaltlich an Schärfe zu verlieren. (s. Abb. 1.3)
 8. [Anfang](#)

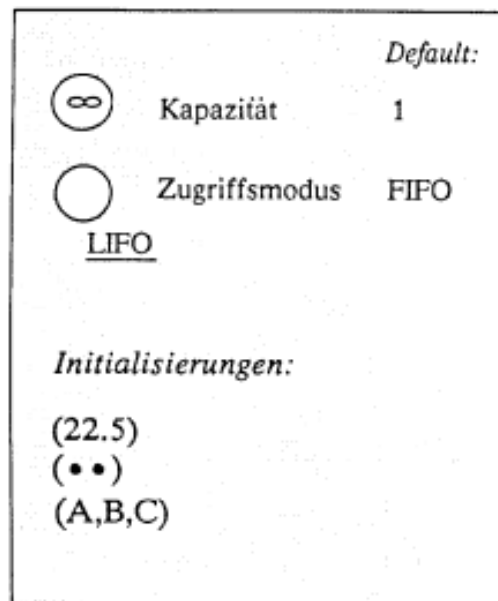
1.1.1 Zusammenfassung der Attribute

Funktionsnetze bauen auf der Kanal-/Instanz-Interpretation von Petrinetzen auf. Durch die Erweiterung der sprachlichen Ausdrucksmittel, z. B. durch Zusammenfassung von Teilnetzen zu eigenen Symbolen, erleichtern Funktionsnetze die Modellierung von Systemen. Die folgende Abbildung zeigt die Attributierung der Funktionsnetze in einer Übersicht:

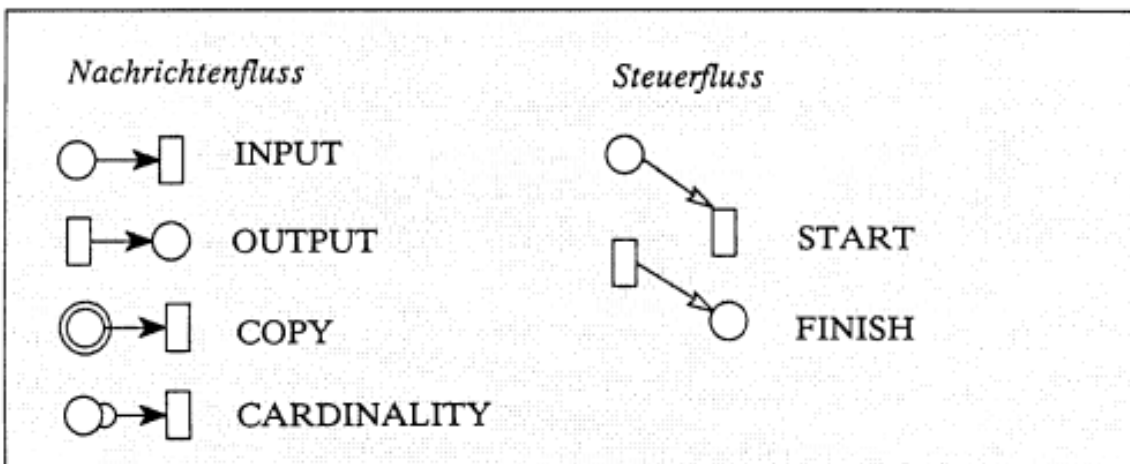
Instanzen:



Kanäle:



Systembeziehungen:



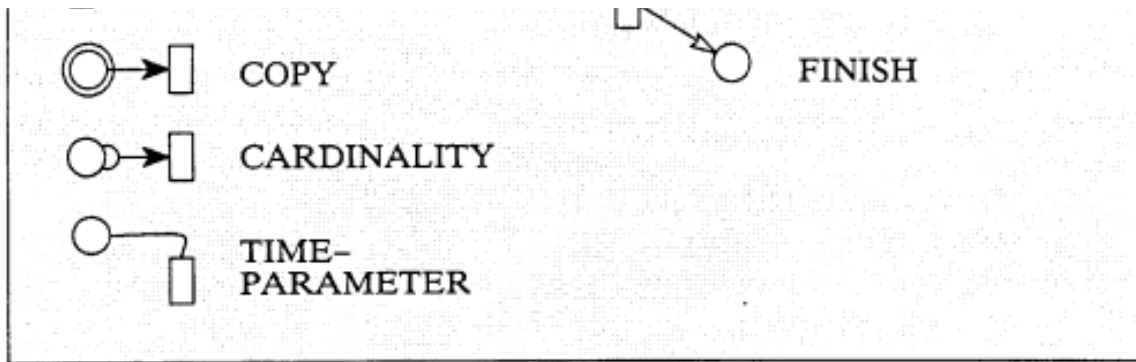


Abbildung 1-1 Zusammenfassung der FN-Attribute

[Anfang](#)

1.1.2 Beispiel: Fibonacci Zahlen

Aufgabe: Modellierung der Produktion der Fibonacci-Zahlen mittels Funktionsnetzen:

Hintergrund: $FIBO(n) = FIBO(0) = 0$, $FIBO(1) = 1$ und $FIBO(n-2) + FIBO(n-1)$, $n > 1$

Lösung:

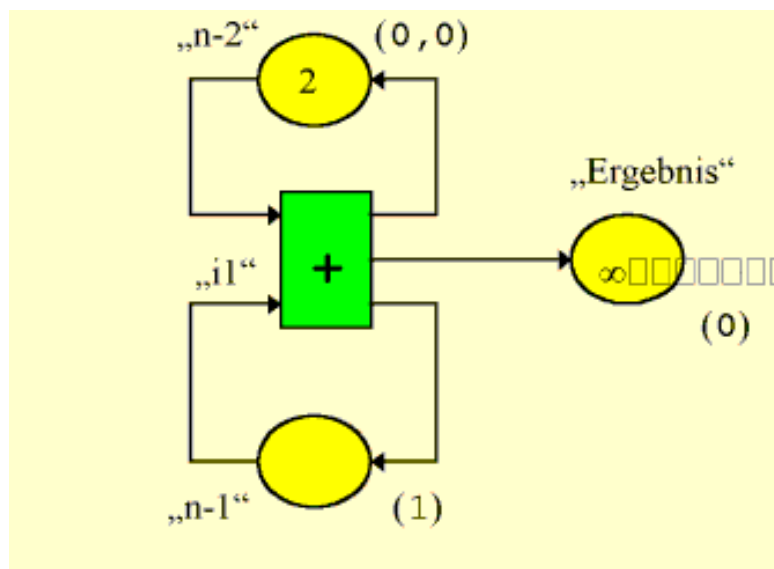


Abbildung 1-2 Produktion von Fibonacci-Zahlen

Die Reihe der Fibonacci-Zahlen wird im Kanal "Ergebnis" gesammelt, dessen Kapazitätsgrenze unendlich ist.. Die Initialisierung ist mit der Zahl 0 erfolgt. Der Kanal "n-2" enthält immer zwei Zahlen in einer FIFO-Schlange. Die Instanz "i1" kann (theoretisch) beliebig oft schalten. Nach dem ersten Durchlauf weisen die Kanäle folgende Inhalte auf: "Ergebnis" eine 0 und 0, "n-1" eine 1, "n-2" eine 0 und eine 1.

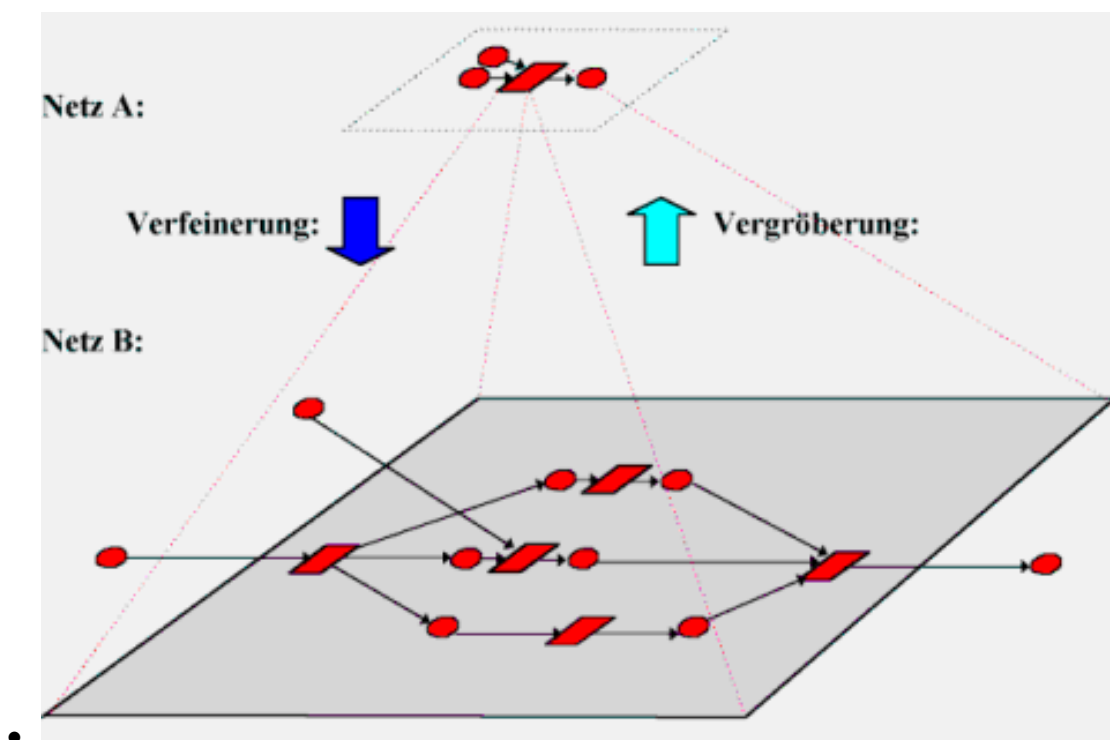
Ergebnis: 0,1,1,2,3,5,8,13,21,.....

[Anfang](#)

1.1.3 Vergrößerung und Verfeinerung

Das zentrale Konzept soll anhand einer Skizze verdeutlicht werden. Man beachte, daß beim Übergang sich nur die Sichtweise ändert, nicht aber unbedingt die inhaltliche Schärfe:

- Einerseits kann damit willentlich ein Wissensverlust verbunden sein,
- andererseits kann Wissen in auf dieser graphischen Ebene nicht mehr sichtbares "Hintergrundwissen" (Stichwort: Instanzenbibliothek) transformiert worden sein.



- **Abbildung 1-3 Verfeinerung und Vergrößerung**

- [Anfang](#)

1.2 Geschichte

Folgende weitere Einrichtungen haben u.a. das Thema aufgegriffen und aktiv Forschungsbeiträge geleistet:

- TU Berlin, FB Informatik, CIS-Gruppe [Go83],
- TU Berlin, Sonderforschungsbereich SFB 203 Rechnergestützte Konstruktionsmodelle im Maschinenwesen, Teilprojekt A5: CAD Qualitätssicherung, [Tr88], [Ra90], [Pa90]
- Universität Dortmund, [We85]

- Universität Bremen, [Sch84]
- Kernforschungszentrum Karlsruhe [Le86]
- TFH Berlin [Go93]

Ausgangspunkt der Entwicklung ist das Interesse an **Petrinetzen**. In der Zeit um 1975 waren Bedingungs-Ereignis-Netze und Transitionsnetze in den USA auf großes akademisches Interesse gestoßen und folglich wurden sie auch in Deutschland populär. Die ersten Arbeiten an der TU Berlin in der CIS-Forschungsgruppe (Computergestützte Informationssysteme) bewegten sich auf der Abstraktionsebene klassischer Programmiersprachen [Go78], d.h. sie stellten einen neuen Versuch dar zur **Programmierung im Kleinen**. Ein Problem dabei war das explosionsartige Wachstum der Größe solcher Netz-Darstellungen, wenn etwas anderes als ein triviales Beispiel modelliert werden sollte.

Die in Skandinavien entwickelte Modellierungsmethode für Informationssysteme "ISAC" wurde in der CIS-Gruppe adoptiert und mit der Petrietz-Notation kompatibel gemacht. Die ISAC-Methode hat ausschließlich eine Modellbildung zum Ziel (vgl. [Wi86]). Ein weiterer Forschungsschwerpunkt war die Entscheidungsunterstützung bei betriebswirtschaftlichen Systemen. Hier wurde u.a. mit **kontinuierlicher Simulation** (.z.B. Dynamo) gearbeitet.

Der Funktionsnetz-Ansatz stellt nun einen Versuch dar, all die oben genannten Konzepte zu integrieren. Dazu wurde auch ein Programmsystem "FUN" entwickelt, das Simulation und **Analyse** (im Sinne der Petrietz-Theorie) ermöglichte.

Als Hardwarebasis konnte nur ein IBM-Großrechner (370 unter VM) genutzt werden, als Programmiersprache stand u.a. SIMULA zur Verfügung, als Ausgabemedium nur ASCII-Terminals und -Drucker.

Rückschau der verschiedenen Projekte in den rund 20 Jahren in Berlin:

TU Berlin I (IBM Mainframe, SIMULA)

- Aufbereitung Histogramm, TRACE
- ANALYSE, Rückführung auf PN
- SIMULATION Soziotechnischer Systeme
(Mix von diskret/ kontinuierlich, Betriebswirtschaft, Protokollverifikation,
- Baukastenprinzip, Fachgebiete
- APPLE II Graphikoberfläche
- Programmgeneratoren (Synthese I)

TU Berlin II (SFB 203 CAD Qualitätssicherung)

- WS-Editor,
- Datenbasis zum Projektmanagement (Bausteinbibliothek)

TFH Berlin (Portierung auf UNIX, Verteilte Systeme)

- Interprozeßkommunikation (intra- bzw. inter-Rechner)
- Migration des Schwerpunkts zur Programmierung im Großen

- Wiederverwendbarkeit, Qualitätssicherung
- Fokus auf Verteilte Systemen
- exemplarisches Anwendungsgebiet Automobilindustrie
- Formulierung einer Workbench
- Schnittstellen zu GUI und Datenbasen

Copyright ©; Godbersen, 1996 [Anfang](#) 29.09.96 1.htm

2 Modelle zur Entscheidungsunterstützung

lokaler Inhalt:

2.1 [Ziele bei der Modellbildung](#)

2.2 [Deskriptive Modelle](#)

2.3 [Operationale Modelle](#)

2.3.1 [Simulationen zur Leistungsvorhersage](#)

2.3.2 [Prototyping \(Mock Up\)](#)

2.4 [Benutzerinteraktion](#)

1. [Trennung von Algorithmus und GUI](#)

[zurück zum Gesamt-Inhaltsverzeichnis](#)

Es wird zunächst auf die Verwendung von Funktionsnetzen zur Modellierung soziotechnischer Systeme eingegangen. Anschließend erfolgt die Diskussion operationaler Modelle. Die dabei erforderlichen Methoden der Benutzerinteraktion über GUI werden skizziert.

[Anfang](#)

2.1 Ziele bei der Modellbildung

Bezüglich der Mächtigkeit der vorliegenden Modellierungskonzeption seien folgende Stichpunkte genannt:

1. die **Modellbildung** und **-Dokumentation** mit einer anschaulichen graphischen Darstellung
2. Operationen auf den Modellen, insbesondere für die **Entscheidungsunterstützung** als (spezielles) Einsatzgebiet von Informationssystemen (IS) und die **Entwurfsunterstützung** als Teil der Erstellung eines (beliebigen) computergestützten IS
3. eine **einheitliche Beschreibung** von Systemen und Prozessen, unterschiedlicher Anwendungsgebiete, Modellierungsziele und Systemgrenzen, einschließlich der **einfachen Auswertung** (bzw. Aufbereitung) der Ergebnisse und **Online-Kommunikation** zwischen Benutzer und Modell während der Durchführung von Berechnungen
4. die **Vorab-Spezifikation unterschiedlicher Sichten** in der Form von branchenspezifischen Aggregaten, Namen oder Einschränkungen, abgestimmt auf die Bedürfnisse unterschiedlicher Benutzergruppen (z.B. Laie, Experte)
5. **Analyse** struktureller und dynamischer Eigenschaften, u.a. mit Mitteln der Petrinetz-Theorie

6. **Gültigkeitsprüfung** der Modellbildung durch den Einsatz computergestützter Analyse- und Simulationsinstrumente, wobei sich beide Aspekte gegenseitig ergänzen
7. die Schnittstelle zu einer abstrakten Maschine **Methoden-** und **Modellbank** und zum Bau von Prototypen.
8. die weitgehend **computergestützte** Durchführung der Modellierung, Analyse und Simulation mit Hilfe von Software-Tools

[Anfang](#)

2.2 Deskriptive Modelle

Funktionsnetze basieren auf Bedingungs/Ereignisnetzen und lassen sich damit implizit zur **Beschreibung** von soziotechnischen Systemen verwenden indem **Präzedenzen** der **Abarbeitungsfolge** aufgezeigt werden. Zu den verwandten Modellansätzen gehört auch der in Skandinavien weit verbreitete Ansatz "ISAC".

Im Bereich deskriptiver Modelle liegen (u.a. bezüglich ISAC) umfangreiche Erfahrungen mit kommerziellen Softwareprojekten vor. /Ge84/. Die Anwendung im Bereich von Protokollen wird in /Bau83/ diskutiert. Weitere Anwendungsbeispiele sind auch in /Go83a/ aufgeführt.

Das folgende Beispiel zeigt eine Beschreibung der Ziele des Funktionsnetz-Ansatzes [Pa90]:

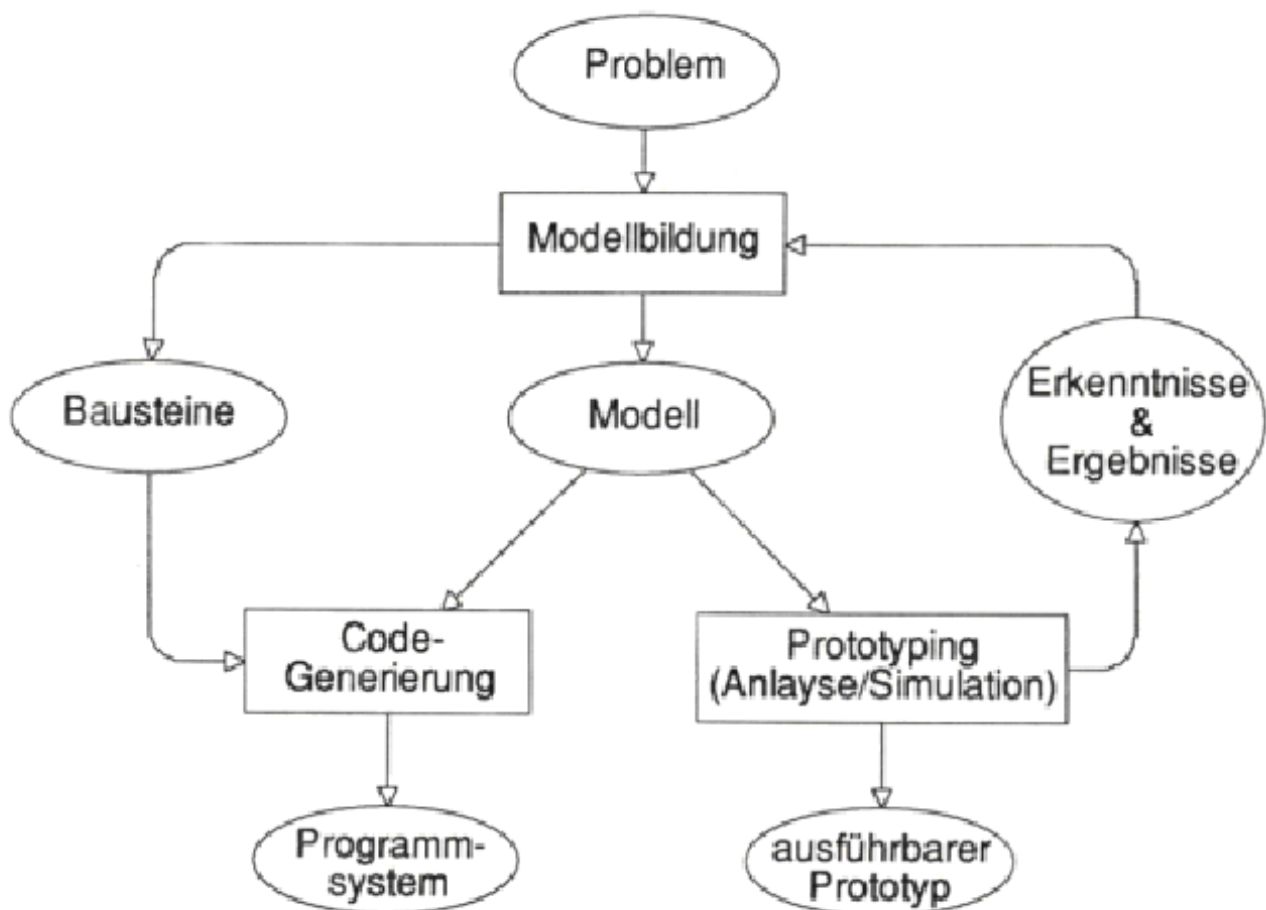


Abbildung 2-1 FN Vorgehensmodell

[Anfang](#)

2.3 Operationale Modelle

Bei operationalen Modellen geht über die Beschreibung von Präzedenzen hinaus. Bei Simulationen zur Leistungsvorhersage werden u.a. Zeitparameter eingesetzt.

[Anfang](#)

2.3.1 Simulationen zur Leistungsvorhersage

Operationale FN sind bisher erfolgreich im Rahmen von Leistungsvorhersagen eingesetzt worden /Sc83/. Es handelt sich um **stochastische Warteschlangenmodelle** zur Untersuchung der Architektur von Multicomputer-Datenbankmaschinen. Dabei werden u.a. Job-Bausteine für die Generierung von Transaktionen und zur statistischen Auswertung benutzt. Weiterhin sind FN auch im Bereich **kontinuierlicher Simulation** einsetzbar /Go83a/.

Es folgt ein Beispiel aus dem Datenbankdesign [Sch86]:

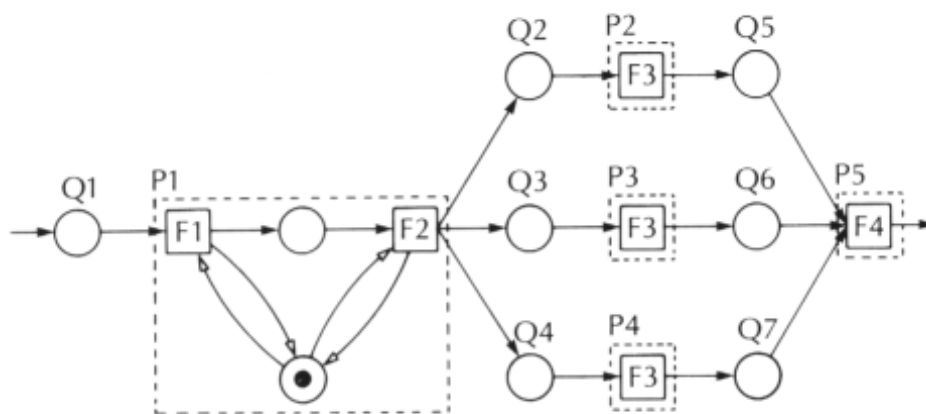


Abbildung 2-2 Paralel processors operation by single instruction, multiple data

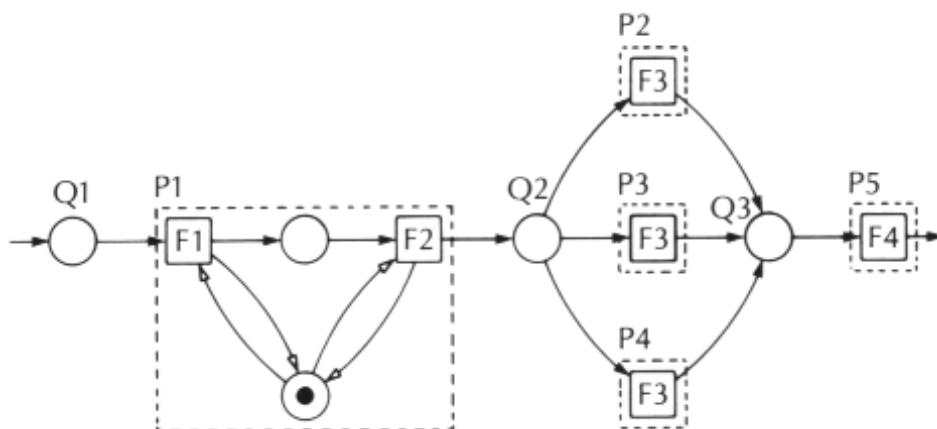


Abbildung 2-3 Paralel processors operation by multiple instruction, multiple data

[Anfang](#)

2.3.2 Prototyping (Mock Up)

Die FN-Spezifikation eines Informationssystems läßt sich zu einem Prototyp weiterentwickeln, der mit geringem Aufwand die Realisierung wichtiger Funktionen des geplanten IS ermöglicht. Die zukünftigen Benutzer werden dadurch in die Lage versetzt, Änderungswünsche einzubringen. Damit wird einerseits eine frühzeitige Fortschreibung der Systemspezifikation ermöglicht.

Darüber hinaus wird es möglich, die in der Job-Bibliothek gesammelten Algorithmen als '**reusable Software**' im späteren Produkt weiter zu verwenden. Diese Vorgehensweise entspricht dem '**evolutionären Prototyping**'.

[Anfang](#)

2.4 Benutzerinteraktion

Zur Handhabung operationaler Modelle muß bereits ein computergestütztes Werkzeug bereitgestellt werden. Wie soll nun die Benutzerinteraktion aussehen?

- **Debug Modus:** Einzelschritt, Hervorhebung der Ausführungsschritte, Abtastpunkte (probe), Halten bei Unterbrechungspunkten (breakpoints)
- **Entwickler Modus:** Visualisierung auf der Ebene der Netzbeschreibung
- **Anwender Modus:** Bedienung ausschließlich über GUI Oberfläche (s.u).

Für den Anwender-Modus ist ein separates GUI einzuführen.

[Anfang](#)

24.1 Trennung von Algorithmus und GUI

In den letzten Jahren haben sich Graphische Benutzerschnittstellen (GUI) weitgehend durchgesetzt. Darüber hinaus werden dafür Programmgeneratoren auf dem Markt angeboten. Es liegt nun Nahe, die Benutzerinteraktion über eine solche Oberfläche abzuwickeln und eine Verbindung mit Netzelementen herzustellen. Beispielsweise in der Form, daß eine Instanz mit der Aufgabe "lese Zahlenwert aus einem GUI-Fenster" bereitgestellt wird.

Das folgende Beispiel zeigt die Trennung (auf der Basis von LabVIEW): Beispiel 2-1 Addierer mit GUI

GUI (MS-Windows):



Schaltplan: (Hier werden Symbole des Programmpakets LabVIEW verwendet)

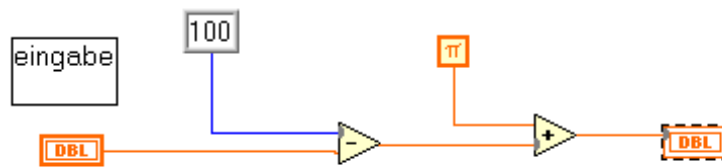


Abbildung 2-4 Addierer aus GUI und Schaltplan

Copyright ©; Godbersen, 1996 [Anfang](#) 29.09.96 2.htm

3 Programmieren im Kleinen

Inhalt:

3.1 [Aufgabenstellung](#)

3.2 [Lösungsansätze](#)

3.2.1 [Aufgabenteilung](#)

3.2.2 [Teilnetze für bedingte Abarbeitung](#)

3.3 [Schnittstellen](#)

3.3.1 [Bausteine](#)

3.3.2 [Modell-Beschreibung](#)

[zurück zum Gesamt-Inhaltsverzeichnis](#)

Es wird die Programmierung von (zunächst kleinen) gebrauchsfertigen Systemen diskutiert, aufbauend auf den bereits im vorherigen Kapitel eingeführten operationalen Modellen.

3.1 Aufgabenstellung

Im letzten Kapitel standen die Modellbildung für Beschreibungszwecke und zur Simulation (z.B. Leistungsvorhersage) sowie Prototypen im Mittelpunkt. Die dabei durchzuführende Programmierung (in diesem Fall die Erstellung von Instanzen-Algorithmen und deren Verknüpfung zu Funktionsnetzen).

Folgende Aufgaben sind zu lösen:

- **Beschränkung** des Einsatzfeldes (hin zu Anwendungsklassen), ggf. auf **Nischen**.
- **Aufgabenteilung** mit anderen bewährten Konzepten. Hier ist insbesondere das GUI und die

- Peripherie von Interesse.
- Reduktion der Komplexität der Modellierung durch geeignete intuitive "Vereinfachungen"
 - Geeignete Abbildung auf **multiple Prozesse** und deren Interprozess-Kommunikation (IPS).
 - Wahl von geeigneten **Datenstruktur-Kontainern** und ggf. einer **Typisierung** der Verkehrsbeziehungen
 - Überlegungen zur **Programmgenerierung** von stand-alone-Lösungen zur Performanz-Verbesserung und einfacheren Distribution.

[Anfang](#)

3.2 Lösungsansätze

Für einige der oben angesprochenen Punkte wird eine Lösung skizziert.

3.2.1 Aufgabenteilung

Die Aufgabenteilung mit GUIs und Peripherie (Dateien, Spreadsheets, Datenbanksystemen, Sensoren, Aktoren) ist sinnvoll. Im Prinzip wird dies erreicht durch eine **Platzhalterfunktion** im Funktionsnetz, wie in der folgenden Abbildung gezeigt:

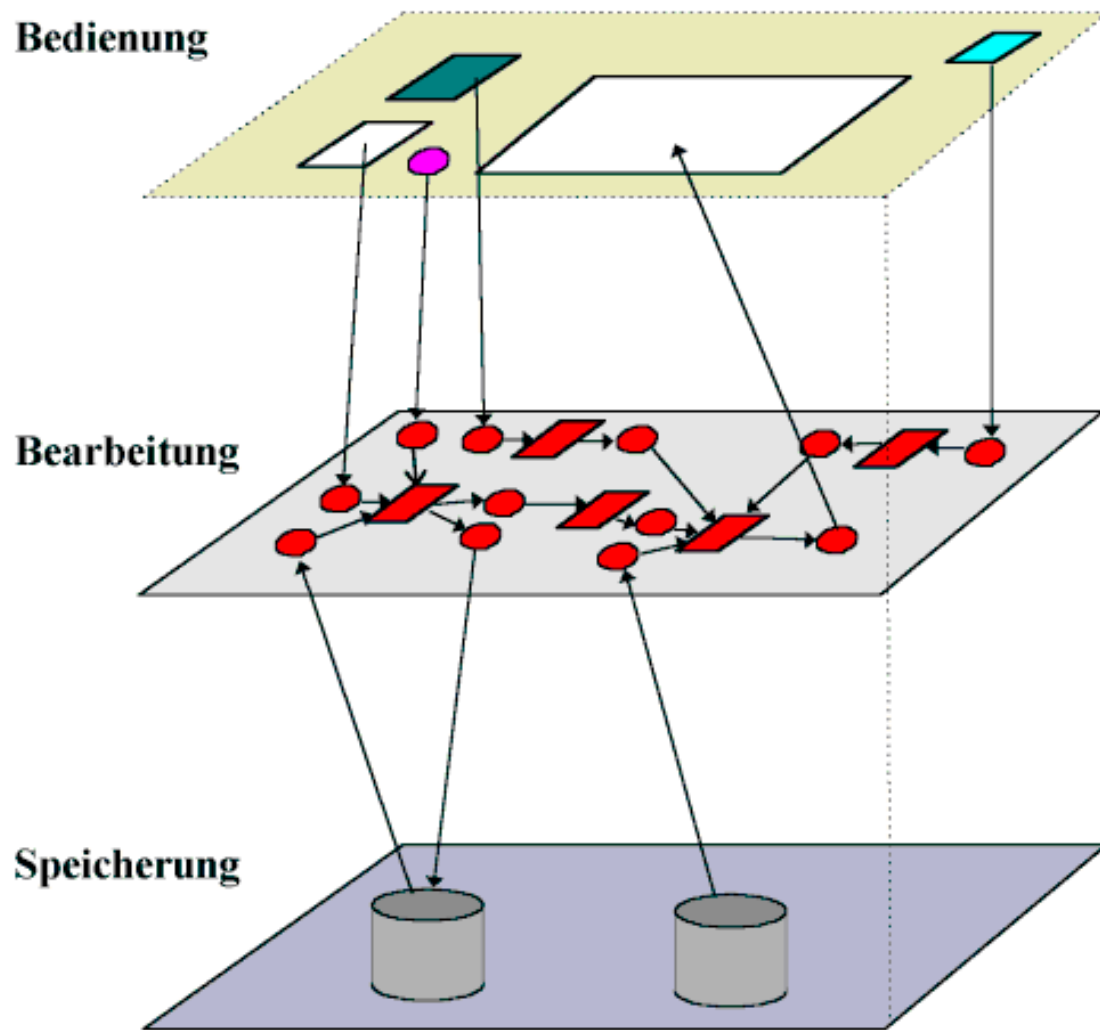


Abbildung 3-1 Aufgabenteilung

Betrachten wir nun die GUI-Programmierung mehr im Detail:

Typischerweise werden **Programmgeneratoren** eingesetzt, die bereits einen vollen Satz von **Prozedurrümpfen** zur Interaktion mit den GUI-Ressourcen bereitstellen.

Damit ist auch automatisch die Abkehr von der klassischen Programmierung hin zur Ereignis-basierten Abarbeitung erfolgt.

Aufgabe der Modellierung im Funktionsnetz ist es dann, die Vielzahl möglicher Ereignisse in so weit wie nötig geordnete Bahnen zu lenken.

Hier besteht noch Klärungsbedarf.

[Anfang](#)

3.2.2 Teilnetze für bedingte Abarbeitung

Zur Reduktion der Komplexität eines Modells oder (-Teils) gehört, sich auf wesentliches zu beschränken, ohne das Informationen für die computergestützte Bearbeitung verloren geht. Aus der Anregung von LabVIEW heraus können z.B. Alternativ- Konstrukte folgendermaßen visualisiert werden (wobei in einer 2D-Darstellung nur jeweils eine Ebene sichtbar ist):

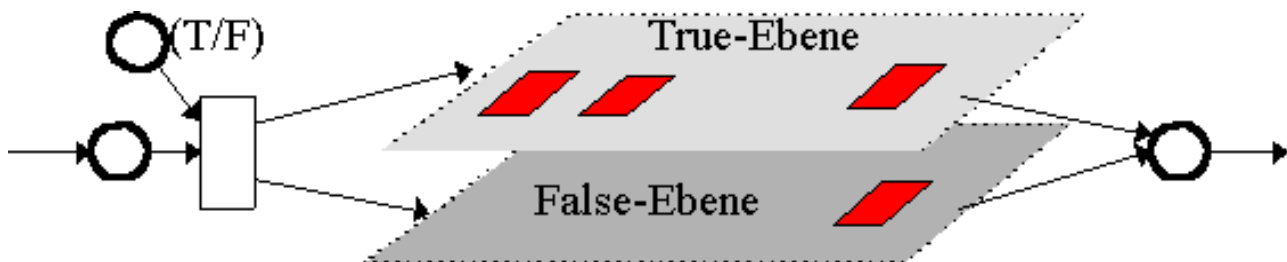


Abbildung 3-2 if/else-Block

Ähnlich ist mit Auswahl- und Wiederholungs-Konstrukten vorzugehen.

[Anfang](#)

3.3 Schnittstellen

3.3.1 Bausteine

Einige Überlegungen zur Gestaltung der Bausteinschnittstelle werden im folgenden festgehalten (s. [Kr91]). Es werden vier Alternativen vorgeschlagen:

1. **ASCII:** Unter UNIX (und unterstützt durch die Programmiersprache C) wurde der Ansatz gewählt, alle Schnittstellen nach einem einheitlichen Schema zu gestalten. Mittels "argc", "argv" und "envp" erhält ein Programm alle wesentlichen Informationen, die es zu seiner Ausführung braucht. Dies erlaubt die Umsetzung eines sehr flexiblen Konzeptes: Bausteine (UNIX-Werkzeuge) sind (fast) beliebig miteinander kombinierbar. Allerdings bedeutet die Einheitlichkeit der Schnittstellentypen (ein String-Vektor) auch erheblichen Mehraufwand innerhalb eines Programms, da die Parameter intern unter dem Aspekt geprüft werden müssen, ob sie mit der Semantik des Programmes verträglich sind. Gegebenenfalls schließt sich dann noch eine Typkonversion an.
Die Wahl von ASCII als Schnittstellenbasis hat sich in Weitverkehrsnetzen sehr bewährt, weil sie einfach, robust und schnell zu definieren ist.
2. **Int, Real & Co.:** Basis-Vorrat von einfachen Datentypen
3. zusätzlich Aggregate (Records, Arrays, etc.)
4. **ASN.1:** Eine weitere Herangehensweise findet sich mit dem "Tag, Length, Value" Ansatz, der ebenfalls weit verbreitet ist (D-Base Dateien, Maskendateien einiger Generatoren, BER etc.). "Tag" ist die Bezeichnung eines folgenden Parameters aus einer vorher definierten Menge von Bezeichnungen. "Length" gibt die Länge des Parameters an (z. B. in Byte), und in "Value" steht der Wert des Parameters selbst.

Für die Gestaltung der Bausteinschnittstelle ist die vierte Variante geeigneter. Zum einen wird durch die Typisierung der Schnittstellen eine Syntaxprüfung bereits zur Editierzeit und nicht erst zur Laufzeit (jedes Mal beim Schalten einer Instanz!) ermöglicht. Zum anderen ist die Verwendung von Parametern, die in ASN.1 spezifiziert und nach BER codiert werden, wegen der ohnehin notwendigen Abbildung auf diese Art der Präsentation von Daten in einem verteilten System gemäß ISO/OSI-Referenzmodell, konsequent. ASN.1 ist als **Spezifikationsprache** geeignet, da es als Hochsprache Daten mit großem Abstraktionsgrad beschreiben kann. ASN.1 ist leicht abbildbar auf Implementierungssprachen wie ADA, C oder Pascal. ASN.1 verfügt außerdem über einen Mechanismus, der auch die Definition von Daten während des Austausches erlaubt und der somit das starre Konzept einer vorgegebenen Typmenge flexibilisiert.

Die Lösungen unterscheiden sich auch bezüglich der Stärke der Typbindung.

[Anfang](#)

3.3.2 Modell-Beschreibung

Zur Archivierung wird eine Beschreibung eines aktuellen Funktionsnetz-Modells benötigt. Die folgenden Ausführungen geben eine Skizze, wie dies geschieht. (Es fehlt die Zuordnung des Layouts für die graphische Repräsentation). Die grafische Darstellung siehe folgendermaßen aus [Jä90].

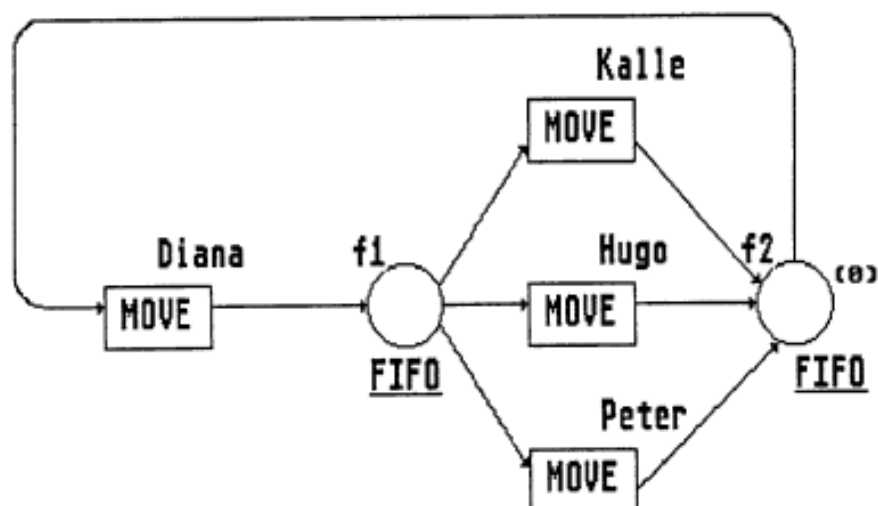


Abbildung 3-3 Einfaches Modell "Diana"

Die Modell-Datei von diana:

```
A G E N C Y (type, name, time_consumption, memory):
```

```
MOVE Hugo 0.0 NoMemo
```

MOVE Peter 0.0 NoMemo

MOVE Kalle 0.0 NoMemo

MOVE Diana 1.0 NoMemo

C H A N N E L (access, type, name, capacity, initial_marking):

FIFO INT f2 1

FIFO INT f1 1

A R C S (agency, channel, type):

Hugo f1 I

Kalle f1 I

Peter f1 I

Diana f1 O

Diana f2 I

Hugo f2 O

Peter f2 O

Kalle f2 O

*fi

Copyright ©; Godbersen, 1996 [Anfang](#) 29.09.96 3.htm

4 Stand der Technik

Inhalt:

[4.1 Visual Programming](#)

[4.1.1 Rapid Application Development \(RAD\)](#)

[4.1.2 Kommerzielle Nischen-Ansätze](#)

[4.1.2.1 Iris Explorer](#)

[4.1.2.2 Ad Oculos](#)

[4.1.2.3 LabVIEW](#)

[4.2 Middleware](#)

[4.2.1 Client/Server Architekturen](#)

[4.2.2 Middleware](#)

[4.2.3 Componentware](#)

[4.2.4 Kommerzielle Vorschläge](#)

[4.2.4.1 ToolTalk](#)

[4.2.4.2 Corba 2.0](#)

[4.2.4.3 OLE](#)

[4.2.4.4 OpenDoc](#)

[zurück zum Gesamt-Inhaltsverzeichnis](#)

Dieses Kapitel befaßt sich mit einigen wichtigen Vorschlägen und Lösungen aus dem Umfeld.

- Im Abschnitt Visual Programming werden allgemeine Konzepte vorgetragen und drei kommerziell angebotene Nischen-Lösungen vorgestellt.
- Der zweite Teil beschäftigt sich mit Middleware: Dienstprogrammssystemen zur Unterstützung der Erstellung von Verteilten Systemen.

Die Quellen der Zitate sind angegeben.

[Anfang](#)

4.1 Visual Programming

Unter diesem Stichwort sammeln sich sowohl kommerzielle erfolgreiche Programmsysteme wie

Visual Basic

als auch universitäre Forschungsansätze.

Ein deutschsprachiges Buch, das einen Teilausschnitt der Entwicklung beleuchtet, ist unter dem Titel "Visuelles Programmieren" erschienen (Autor: J. Paswig, Hanser Verlag, 1996) [Pa96].

Exkurs: Quelle: [Va94]

A future programmer may look less like a writer laying down words and more like an electrician wiring together circuit components

With the arrival of programming tools such as Visual Basic, the term visual programming has entered the lexicon of mainstream computing. However, the term means very different things to different people. For many, Visual Basic--and its cousin, Visual C++--may be all they know, or expect to know, about visual programming. To others, Visual Basic is hardly representative of the vast array of different technologies that share the label of visual. Among the other alternatives are Digital's Parts, Powersoft's PowerBuilder, Meta Software's Design/CPN, and Novell's Visual AppBuilder.

The goal of all of these different approaches is the same: to make programming easier for programmers and accessible to nonprogrammers. Some are used for rapid prototyping and rapid applications development, others are used for systems or applications design, and still others can produce stand-alone applications for distribution. In all cases, visual languages and visual programming let users put more effort into solving their particular problem rather than learning about the computer or about a programming language. This way, an engineer, for example, does not also have to be a computer programmer to simulate a complex control system.

[Anfang](#)

4.1.1 Rapid Application Development (RAD)

Exkurs: Quelle: [Li95]

RAD (rapid application development) tools promise two advantages over traditional programming.

- The first advantage is a **shorter, more flexible development cycle**, enabling you to leap directly from prototype to finished application.
- The second advantage is that a reasonably sophisticated **end user can develop applications**.

RAD tools often require you to write code. But if you use them properly, you can reduce many programming tasks to drag-and-drop simplicity.

The roots of RAD lie in the **prototyping tools** of yore. With such tools, developers could quickly **mock up** an application so the end user could see and experience it before the design was finalized. Prototypes were the ultimate design tool, because they virtually eliminated misunderstandings about an application's look, feel, and capabilities. Once the developer and end user agreed on a prototype, the developer simply created an application that looked and acted like the prototype.

But these prototyping tools usually provided only "smoke and mirrors" for the developer. Prototypes rarely became final applications. Developers were building the application twice. To solve this problem, RAD tools extend the capabilities of prototyping tools by providing developers with everything they need to

- build a **prototype** as well as
- turn it into a **fully functional application**.

It's a fairly elegant solution. Developers build applications with RAD tools **primarily by designing the interface**. They assemble components such as buttons, menus, data windows, and combo boxes. Developers are more concerned with what the program does than they are with how it does it. They show the application to users, get feedback, and make modifications to the application. This process continues until the user is happy. The time gained from using a RAD tool can be immense.

Most IBM/VisualAge programmers report the ability to create up to 80 percent of an application visually, with the last 20 percent consisting of specialized functions.

[Anfang](#)

4.1.2 Kommerzielle Nischen-Ansätze

Es werden drei kommerziell vertriebene Pakete kurz vorgestellt, die zwar jeweils für Spezialgebiete eingesetzt werden, aber sich auf mit Funktionsnetzen enthaltene Konzepte beziehen und darüber hinaus auch Anregungen für eine Fortschreibung geben.

[Anfang](#)

4.1.2.1 Iris Explorer

Ein Visualisierungs-Werkzeugkasten, derzeit auf Silicon Graphics WS zur Verfügung stehend, früher eine Entwicklung aus dem US-amerikanischen Hochschulbereich (apE) [SG91], [La95]. Der IRIS Explorer ist ein System, mit dem man leistungsfähige **Datenvisualisierungs**-Applikationen erstellen kann.

Der Explorer ist ein sehr komplexes System. Er besteht aus mehreren Werkzeugen. Die Benutzung der einzelnen Werkzeuge erfordert unterschiedliche Systemkenntnisse, deshalb hat man den IRIS Explorer in Benutzerebenen eingeteilt. Die unterste Ebene, die das gesamte Potential des Systems zugänglich macht, wird oftmals durch Programmierer oder andere Anwender mit Programmierkenntnissen benutzt. Je höher die Ebene in der Rangordnung steht, desto einfacher ist ihre Bedienung. Aber auch die Beschränkungen für den Anwender nehmen von Ebene zu Ebene zu. Die genaue Einteilung sieht folgendermaßen aus:

- Ebene 1 Benutzung bestehender Applikationen,
- Ebene 2 Erstellung von Applikationsprototypen,
- Ebene 3 Erstellung von Prototypen neuer Module,
- Ebene 4 Schreiben neuer Module.

In Ebene 1 arbeiten Anwender, die vorhandene Applikationen benutzen und diese an ihre eigenen Anforderungen anpassen. In Ebene 2 erstellt man neue Applikationen. In beiden Ebenen arbeitet man mit dem Map Editor. In der 3. Ebene kann man mit einer C-ähnlichen Sprache, "LatFunction's Shape" genannt, neue Funktionalität in das System einbringen. In Ebene 4 arbeitet man mit den Werkzeugen Module Builder und Data Scribe.

Der Hersteller hat schon eine beträchtliche Anzahl von Programmen, die in Form von Modulen vorliegen, in das System integriert. Somit kann man ohne größere Vorbereitungen mit der Erstellung von Applikationen, die Maps genannt werden, beginnen. Der Explorer verfügt zu diesem Zweck über Werkzeuge, mit denen man nicht nur Applikationen erstellen kann, sondern diese auch seinen eigenen Anforderungen anpassen kann. Grundlage ist die Modulbibliothek, die alle Module enthält. Der Map Editor ist die Arbeitsplattform, auf der man einzelne Module mit interaktiven, visuellen Mechanismen zu Maps zusammenlinkt. Der Group Editor und der Parameter Function Editor sind die Werkzeuge, mit denen man die Maps seinen eigenen Wünschen anpaßt. Ein robustes System von Datentypen dient zur Informationsübertragung zwischen den Modulen. Ein weiteres leistungsfähiges Werkzeug des Explorers ist der Module Builder, mit dem man benutzerdefinierte Funktionen in Module einbindet und somit die Funktionalität des Explorer-Systems erhöhen kann. Das Data Scribe ist das dritte bedeutende Werkzeug neben Map Editor und Module Builder. Es dient zum Transformieren und Filtern von Daten.

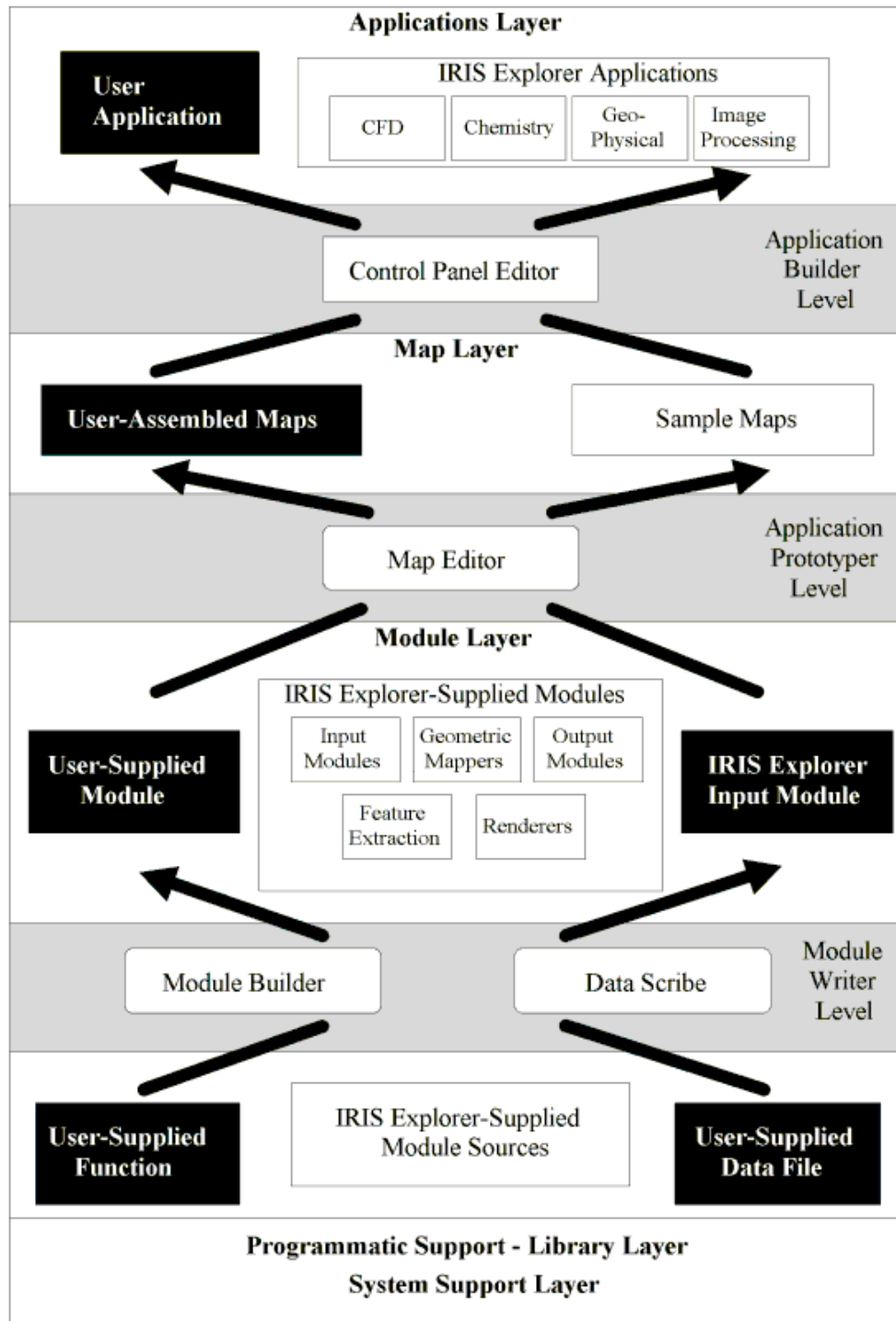


Abbildung 4-1 Benutzerebenen des IRIS Explorers

Die obige Abbildung zeigt die Benutzerebenen und die Werkzeuge, mit denen man innerhalb der Ebenen arbeiten kann. Die Pfeile zeigen die Entstehung neuer Module beziehungsweise neuer Applikationen an. Alle Gruppen, die durch schwarze Kästchen gekennzeichnet sind, können vom Benutzer beeinflusst werden.

Die IRIS Explorer-Komponenten im Überblick Explorer-Datentypen

Der Grund für die Benutzung spezieller Datentypen besteht in der Notwendigkeit, zur Laufzeit einer Applikation genauen Aufschluß über die interne Struktur der Daten zu haben. Zwei Beispiele sollen das verdeutlichen.

1. Bei der Herstellung eines Links zwischen zwei Modulen wird der Map Editor prüfen, ob die Datentypen der gewünschten Verbindung **zusammenpassen**, bevor eine Operation auf den Datentypen erfolgt.
2. Wenn eine Verbindung über die Maschinengrenze hinaus erfolgt, wird das System diese Datentypen in ein

maschinenunabhängiges Format umwandeln, damit stets eine korrekte Interpretation der Daten stattfindet.

Der IRIS Explorer verfügt über fünf ihm bekannte Datentypen, die sich aus skalaren Datentypen zusammensetzen. Die Datentypen sind wie folgt benannt:

- Lattice,
- Pyramid, Parameter, Geometry und
- Unknown.

Der **Lattice (Gitter)**-Datentyp ist der am häufigsten benutzte Datentyp innerhalb des Systems. Deshalb soll er hier genauer beschrieben werden. Er besteht aus einer Struktur, die zwei multidimensionale Felder enthält - den Datenspeicher und den Koordinatenspeicher. Der Datenspeicher enthält die Daten jedes Knotens im Gitter. Der Koordinatenspeicher gibt die Lage der Knoten im n-dimensionalen Raum an. Beide Speicher werden separat abgelegt, so daß es möglich ist, mehrere Gitter mit verschiedenen Daten auf die selben Koordinaten zu projizieren.

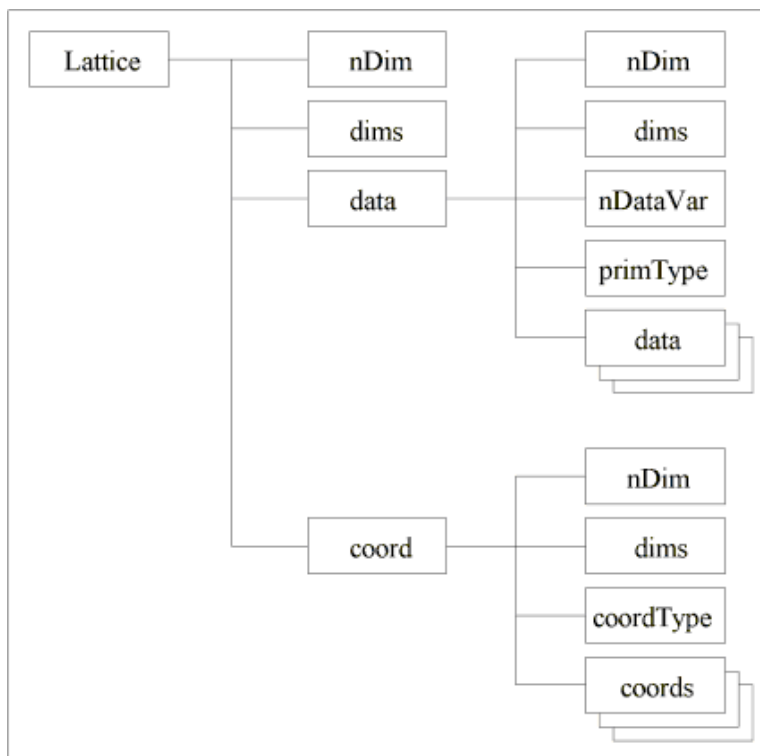


Abbildung 4-2 Der Datentyp Lattice

Erläuterung der Begriffe:

nDim Dimension,

dim Vektor, der die Ausmaße pro Dimension enthält,

nDataVar Anzahl der Variablen pro Knoten,

primType skalarer Variablentyp (float, byte, integer,...),

coordType Typ des Koordinatentyps (uniform, perimeter, curvilinear),

nCoordVar Anzahl der Koordinaten pro Knoten (nur bei curvilinear-Koordinatentyp)

Die verschiedenen Koordinatentypen ermöglichen unterschiedliche Darstellungsformen der Gitterstruktur. Bilder sind im allgemeinen im uniform-Koordinatentyp abgelegt. Dabei sind die Abstände aller Knoten im Gitter gleich. Die Ausmaße des Gitters werden durch eine Bounding Box vorgegeben. Der perimeter-Koordinatentyp ermöglicht die Darstellung rechteckiger Gitter, bei denen die Abstände zwischen den Knoten nicht identisch sind. Mit dem curvilinear-Koordinatentyp kann man nichtrechteckige Gitter erzeugen. Hier werden die Koordinaten jedes Knotens explizit angegeben.

Der Aufbau einer einfachen Gitterstruktur wird an folgendem Beispiel deutlich. Für die Präsentation einer Colormap im Lattice-Datentyp benötigt man ein 1-dimensionales Gitter mit vier Variablen pro Knoten als Datenspeicher. Die Variablen pro Knoten liegen im Float-Format vor. Der Koordinatentyp ist im allgemeinen uniform. Colormap im Lattice-Format:

```
nDim 1
dims 256
nDataVar 4 (red, green, blue, alpha)
primType float
coordType any
```

Module und Modulbibliothek

Module sind aus der Sicht des Betriebssystems **Prozesse**. Um mit anderen Modulen kommunizieren zu können, haben Module Schnittstellen - die Input- und Output-ports. Jeder Modul kann bis zu zwanzig Input- und zwanzig Outputports haben. Die Inputports teilen sich in benötigte Ports und optionale Ports. Nur wenn die benötigten Ports einen Link zu anderen Modulen besitzen oder eine Benutzereingabe entgegennehmen, kann der Modul ausgeführt (er feuert) werden.

Die vom Hersteller mitgelieferte Modul Gruppen sind:

- Import von Daten,
- Numerische Analyse,
- Bildverarbeitung,
- Analyse spezieller Teile von Strukturen,
- geometrische Repräsentation von Daten,
- Rendering,
- Export von Daten.

Alle Module und Maps sind in der Modulbibliothek abgelegt. Dem Benutzer steht die Möglichkeit offen, die Module nach eigenen Wünschen in Gruppen zu unterteilen. So kann man Gruppen erzeugen, die der genannten Einteilung entsprechen.

Wie oben erwähnt, sind Module ausführbare Programme. Um sie in das Explorer-System einzugliedern, bestehen die Programme aus drei Schichten.

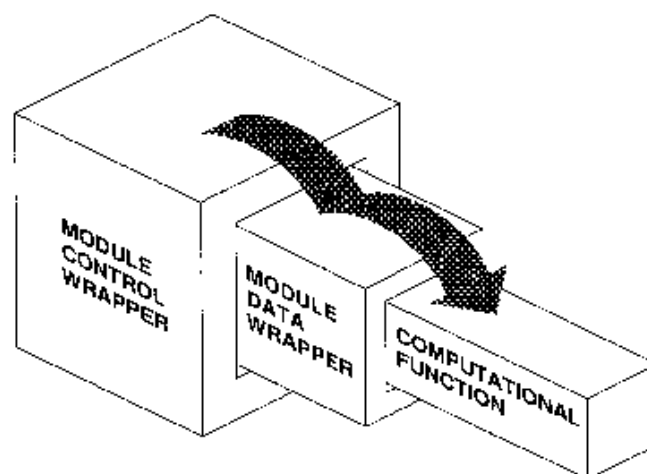


Abbildung 4-3 Die interne Struktur eines Moduls

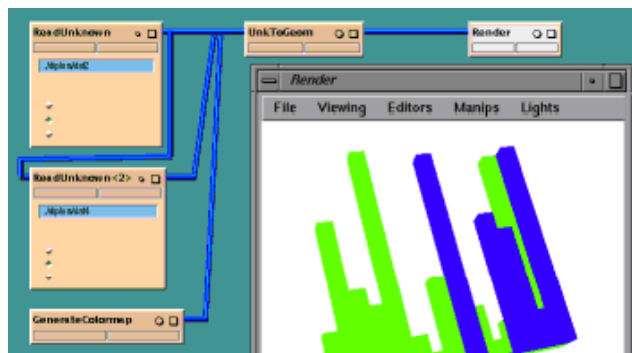
Für die Benutzer, die aus dem Reservoir von Modulen Applikationen erstellen, sind diese Schichten nicht sichtbar. Für Anwender, die neue Module erstellen oder vorhandene Module verändern, ist die interne Struktur eines Moduls von großem Interesse.

Der Module **Control Wrapper** (MCW) hat folgende Aufgaben:

- Kommunikation mit anderen Modulen und dem User Interface (Map Editor),
- Management der Daten an den Ports,
- Überwachung des Ausführungsalgorithmus' des Moduls.

Der Module **Data Wrapper** (MDW) ist verantwortlich für:

- die Konvertierung der Daten an den Ports in die Daten der benutzerdefinierten Funktion und umgekehrt,
- Zerlegung der komplizierten Explorer-Strukturen in seine Bestandteile. Als Maps bezeichnet man die Explorer-Netzwerke, die man aus den Modulen erstellt. Eine Map ist eine eigenständige Applikation. Sie besteht aus den in ihr befindlichen Modulen und den Links (Verbindungen) zwischen den Modulen.



• Abbildung 4-4 Map

Der **Map Editor** ist das User Interface, mit dem man Maps erstellen oder verändern kann. Hier stellt der Benutzer die Module zu einer Map zusammen, erzeugt die Links zwischen den Modulen und stellt die Modulparameter ein.

Der **Module Builder** ist ein weiteres Werkzeug des Explorers. Nur mit diesem Werkzeug ist es möglich, neue Funktionalität, die nicht auf dem Lattice-Datentyp basiert, in das System einzubringen. Der Module Builder ist ein visuell unterstützter Codegenerator. Er ist ein Werkzeug, das drei Aufgaben erfüllt:

1. Erstellen der Modulressourcendatei,
2. Generieren des Wrapper-Codes des Moduls,
3. Generieren eines Makefiles und Erstellung der Module.

Die Modulressourcendatei enthält alle Informationen, die benötigt werden, um von anderen Modulen erkannt zu werden. Das Makefile wird für das Kompilieren und das Linken eines Programms, sowie für die Einbindung in die Explorermgebung verwandt.

Die Vorgehensweise, um einen neuen Modul zu erstellen, schließt folgende Schritte ein:

- die Definition der Input- und Outputports,
- die Definition der Widgets, die von den Ports benutzt werden,
- die Definition des Aufrufs der benutzerdefinierten Funktion,
- die Assoziation von Elementen aus den Explorer-Datenstrukturen mit den Funktionsargumenten der benutzerdefinierten Funktion,
- den Aufbau der Widgets,
- das Erstellen des Moduls mittels Makefile.

Mit dem **Parameter Function Editor** kann man eine Relation zwischen den Parametern eines Moduls herstellen. Er kann nur aus dem Map Editor aufgerufen werden. Mit diesem Werkzeug kann man selbst Inputports mit Daten versorgen. Diese Daten sind von anderen Inputparametern abhängig. Die Operationen umfassen mathematische, logische und auswählende (if..then..else) Funktionen.

Der **Control Panel Editor** ist das Werkzeug zum Gestalten des Aussehens der Module. Er kann aus dem Map Editor, dem Module Builder und dem Data Scribe aufgerufen werden. Wenn dem Benutzer die Eingabe von Werten ermöglicht werden soll oder der Modul eine Datenausgabe in einem Fenster innerhalb des Moduls (siehe Render-Modul) realisieren soll, dann kann man mit dem Control Panel Editor das Aussehen des Moduls und die Initialisierungswerte der Ports definieren.

Das Ausführungsmodell des IRIS Explorers Verteilter, dezentralisierter Datenfluß

Der IRIS Explorer ist ein MVS (Modular Visualization System). Er ist so konzipiert, daß die Module in einer Map auf verschiedenen Workstations laufen können. Die Ausführung einer Map basiert auf einem verteilten, dezentralisierten Datenfluß-modell. Dieses Modell kommt zur Anwendung, wenn Module in einer Map feuern und dabei neue Daten an ihren Outputports produzieren.

Die typische Hardware in Rechnernetzen besteht aus Workstations verschiedener Hersteller. Der IRIS Explorer arbeitet auf allen Maschinen in heterogenen Netz-werken, die mit dem Betriebssystem UNIX, dem X Window-System und Motif ausgestattet sind.

Module in einem IRIS Explorer Netzwerk (Map), sind UNIX-Prozesse. Sie kommunizieren über Pipes und Sockets miteinander. Pipes und Sockets sind Bestandteile des UNIX-Systems, mit denen Prozesse untereinander Daten austauschen können. "Sockets sind Kommunikationsendpunkte. Ähnlich wie ein Telefonhörer das Endstück der Telefonleitung darstellt und menschlicher Kommunikation dient, stellt ein Socket einen Endpunkt für die Kommunikation zwischen Programmen dar. Ein Socket kann für beide Richtungen der Kommunikation genutzt werden." ([Deter89] Seite 408). Die Prozeßkommunikation kann auch über die Rechnergrenzen hinaus erfolgen. Pipes sind "Einbahnstraßen" zwischen Prozessen, auf denen Daten fließen.

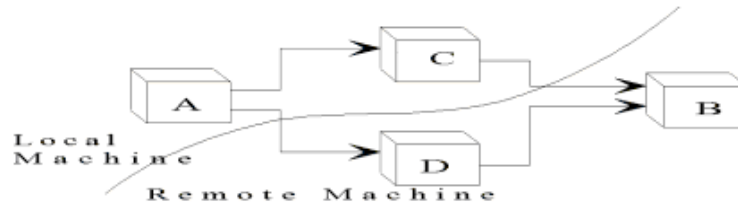


Abbildung 4-5: Ein einfaches Explorer-Netzwerk

Jeder Modul kann ausgeführt werden (er feuert), wenn sich an einem Inputport die Daten verändern, oder wenn man ihn über ein Menü explizit dazu auffordert, zu feuern. Module ohne Inputports müssen immer zum Feuern aufgefordert werden.

Der Modul A feuert, wenn an einem Widget des Moduls ein neuer Wert eingestellt wird, oder wenn man ihn zum Feuern auffordert. Dadurch wird die benutzer-spezifizierte Funktion ausgeführt und ein neues Ergebnis an den Outputport des Moduls geschickt. Dieses Ergebnis wird an die Inputports der Module C und D weitergeleitet. Jetzt feuern auch diese Module. Der Modul B wird jetzt zweimal feuern, da an seinen beiden Inputport neue Daten anliegen.

Prozeßorganisation

Das Graphical User Interface (GUI) ist die Oberfläche, mit der der Benutzer arbeitet. Im Explorer-System ist das der Map Editor. Durch die Arbeit mit dem Map Editor werden Informationen über die Benutzung bestimmter Datentypen und den Austausch der Daten zwischen Modulen aufgebaut. Diese Informationen erhält der Global Communication Server (GC). Er ist für das gesamte Management der Map verantwortlich. Pro Sitzung mit dem IRIS Explorer existiert genau ein GC. Der GC befindet sich auf der Maschine, auf der der Anwender den Explorer gestartet hat. Er berechnet und erstellt den Graphen der Map. Er ist der "Bevollmächtigte", der die Veränderungen der Topologie des Netzes und die Einstellungen der Parameter einer Map an die Local Communication Server (LC) weitergibt. Die LCs existieren auf jeder Maschine, von der der Benutzer Module zu seiner Map hinzugelinkt hat. Ein LC verwaltet alle Module einer Map, die auf seiner Maschine laufen.

Der LC ist verantwortlich für:

- das Feuern von lokalen Modulen,
- den Aufbau von Pipes und Sockets,
- die Aufteilung des Speichers,
- die Übergabe von solchen Daten an den GC, die für den Benutzer bestimmt sind,
- die Zerstörung von Modulen.

Kommunikationsmodell

Bei der Programmierung des IRIS Explorers haben die Entwickler versucht, den Datenfluß optimal zu gestalten. Innerhalb des Explorers gibt es zwei Methoden der Kommunikation - die direkte und die indirekte Methode.

Die Kommunikation zwischen dem Benutzer (über das User Interface) mit den Modulen verläuft auf **indirektem Weg**. Wenn der Benutzer im Map Editor einen Parameter eines Moduls ändert, sendet dieser den neuen Wert an den GC, dieser an den LC, der den Modul verwaltet. Der LC sendet die Nachricht an den Modul. Die indirekte Methode der Kommunikation ist für geringe Datenmengen geeignet. Hier werden nur Eingaben vom Benutzer verarbeitet. Diese Informationen werden in sehr kleine Pakete

zerlegt, die in kurzen Zeitabständen gesendet werden. Eine Ausnahme bildet der Modul **Render**. Hier kommuniziert der Benutzer direkt mit dem Modul.

Im Gegensatz zur indirekten Methode findet die Kommunikation zwischen Modulen auf **direktem Weg** statt. Die Menge der zu übertragenden Daten kann sehr groß sein. Deshalb ist die Zerlegung in sehr kleine Pakete und die Verschickung über den LC nicht geeignet. Bei der Kommunikation zwischen Modulen auf der gleichen Maschine sind die verschickten Datenmengen noch sehr klein. Hier werden nur Referenzen auf Daten ausgetauscht, da sie sich den Speicher der Maschine teilen. Beide greifen damit auf das selbe Objekt zu. Bei der Kommunikation zwischen Modulen unterschiedlicher Maschinen muß das gesamte Datenobjekt verschickt werden.

Auch die Kommunikation zwischen Modulen auf Maschinen mit unterschiedlicher Datenrepräsentation ist so möglich. Die Daten werden vor der Verschickung vom MCW des sendenden Moduls auf ein Netzwerk-neutrales Format konvertiert.

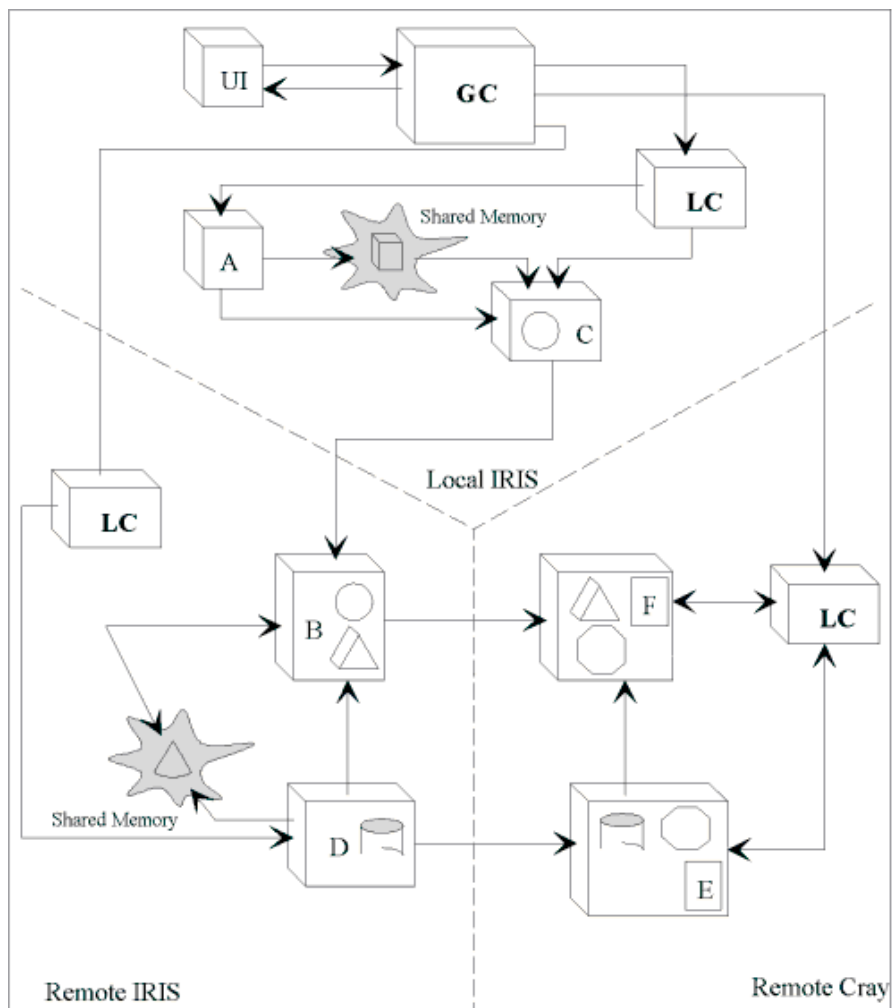


Abbildung 4-6: Kommunikation im Explorer-System

[Anfang](#)

4.1.2.2 Ad Oculos

Preisgekrönte Bildverarbeitungssoftware aus Deutschland. Referenz: [Bä93] (Bässmann & Besslich: Bildverarbeitung. Ad Oculos. Springer Verlag. 2. Auflage. 1993)

[Anfang](#)

4.1.2.3 LabVIEW

LabVIEW, Akronym für: **L**aboratory **V**irtual **I**nstrument **E**ngineering **W**orkbench der Fa. National Instruments.

Die folgenden Beispiele sollen die Nähe zu den Funktionsnetzen dokumentieren. Für eine tiefere Beschreibung sei auf verwiesen auf:[We95] (L.Wells: LabVIEW Student Edition User's Guide. Prentice Hall, 1995).

Vorgestellt wird das Produkt mit dem Ziel, Ähnlichkeiten mit dem Funktionsnetz-Ansatz herauszuarbeiten. Für eine vollständige Einführung sei auf die Literatur verwiesen.

How Does LabVIEW Work?

LabVIEW programs are called **virtual instruments (VIs)** because their appearance and operation imitate actual instruments. However, behind the scenes they are analogous to main programs, functions, and subroutines from popular programming languages like C or BASIC. VIs have both an interactive user interface and a source code equivalent, and you can pass data between them. A VI has three main parts:

1. The **front panel** is the interactive user interface of a VI, so named because it simulates the panel of a physical instrument. The front panel can contain knobs, push buttons, graphs, and many other **controls** (user inputs) and **indicators** (program outputs). You input data using a mouse and keyboard, and then view the results produced by your program on the screen.
2. The **block diagram** is the VI's source code, constructed in LabVIEW's graphical programming language, G.. The block diagram, pictorial though it appears, is the actual executable program. The components of a block diagram, **icons**, represent lower-level VIs, built-in functions, and program control structures. You draw wires to connect the icons together, indicating the flow of data in the block diagram.
3. The **icon** and **connector** of a VI allow other VIs to pass data to the VI. The icon represents a VI in the block diagram of another VI. The connector defines the inputs and outputs of the VI. VIs are **hierarchical** and **modular**. You can use them as top-level programs, as subprograms within other programs, or even within other subprograms. A VI used within another VI, analogous to a subroutine, is called a subVI.

With these features, LabVIEW promotes the concept of **modular programming**. First, you divide an application into a series of simple subtasks. Next, you build a VI to accomplish each subtask and then combine those VIs on a top-level block diagram to complete the larger task. Modular programming is a plus because you can execute each subVI by itself, making debugging easy. Furthermore, many low-level subVIs often perform tasks common to several applications and can be used independently by each individual application.

By creating subVIs, you can make your block diagrams modular. This modularity makes VIs **easy to debug, understand, and maintain**.

LabVIEW's power lies in the **hierarchical nature** of its VIs. After you create a VI, you can use it as a subVI in the block diagram of a higher-level VI, and you can have an essentially unlimited number of layers in the hierarchy.

The icon/connector provides the graphical representation and **parameter definitions** needed if you want to use a VI as a subroutine or function in the block diagrams of other VIs. The icon graphically represents the VI in the block diagram of other VIs. The connector terminals determine where you must wire the inputs and outputs on the icon. These terminals are analogous to parameters of a subroutine or function. They correspond to the controls and indicators on the front panel of the VI. The icon sits on top of the connector pattern. The icon and connector of the Temperature Status subVI are shown (here 3 input, 3 output).

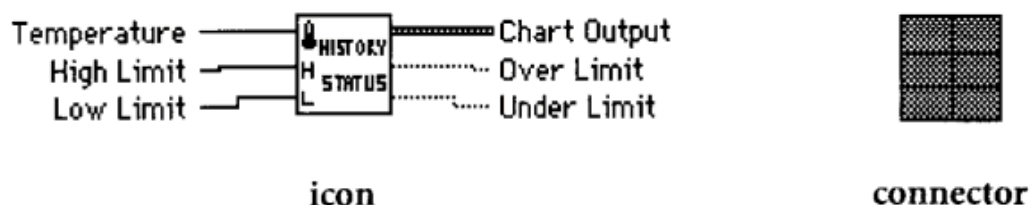


Abbildung 4-7 Ein Icon und der dazugehörige Connector

Datentypen

Die Basis-Datentypen sind **Number**, **Boolean** und **String**. Diese können zu 1D- und 2D-Arrays gebündelt sein. Viele der Operationen

(z.B. add()) sind **polymorph (vielgestaltig)**: sie vertragen sowohl Skalare als auch Felder.

Weiterhin steht die Bündelung beliebiger Daten zu Records unter dem Begriff **Cluster** zur Verfügung. Die Subkomponenten werden über eine laufende Nummer angesprochen. Cluster werden aus Einzeldatenstrukturen mittels der Funktionen "bundle" zusammengestellt und mit "unbundle" wieder zerlegt.

Programmausführung

- **Debug Modus:** single step, execution highlighting, probe, breakpoints
- **Normaler Modus:** Vollständiger Durchlauf, Bedienung über Benutzerinterface. Interpretation der VI's.
- **Compilierter Modus:** Compilierung eines Vis

Beispiele

Die folgenden zwei Beispiele sollen einen Eindruck geben, wie mächtig das Konzept ist.

- Zunächst wird eine Hierarchie von Vis untersucht. Der Baustein "floor" enthält dabei u.a. if_else-Konstrukte (die in zwei Bildern aufgeschlüsselt werden). "floor" wird in "remainder" benutzt. Das Benutzerinterface von "remainder" wird gezeigt.
- Das zweite Beispiel "incremental filter" zeigt einen Container, in dem eine einfache Programmierung mit Parametern und lokalen Variablen realisiert ist.

Beispiel 4-1 Hierarchie mit "floor" und "remainder"

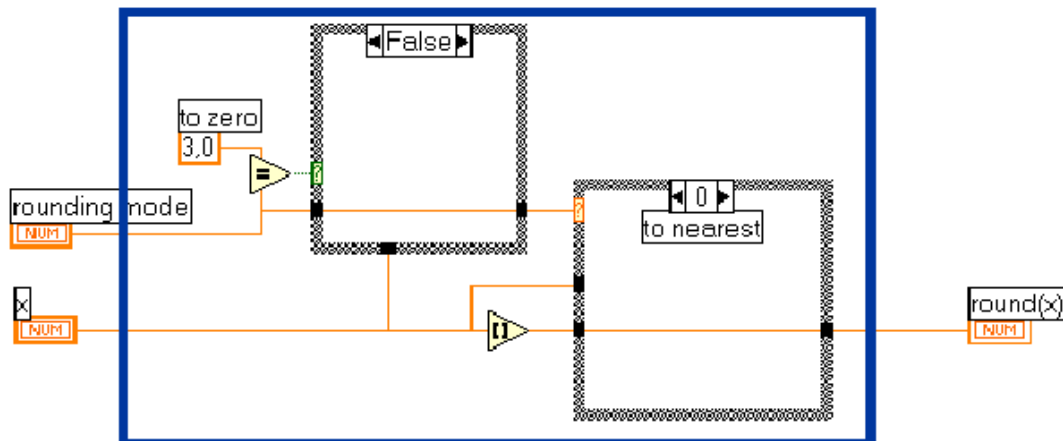


Abbildung 4-8 Die Funktion "floor", Schnappschuss Nr. 1

In blau eingerahmt: zukünftiges Aggregat. Schnappschuss: if_then_else() auf False, case() auf 0. (neue) Funktionen: equal(), case(), round_to_nearest(), if_then_else(), Controls und Indicators in rot. SW-Pfad: examples/apps/tempsys.llb/floor.vi

Die selbe Funktion, nur mit einem anderen Schnappschuss: if_then_else() auf True, case() auf 0.

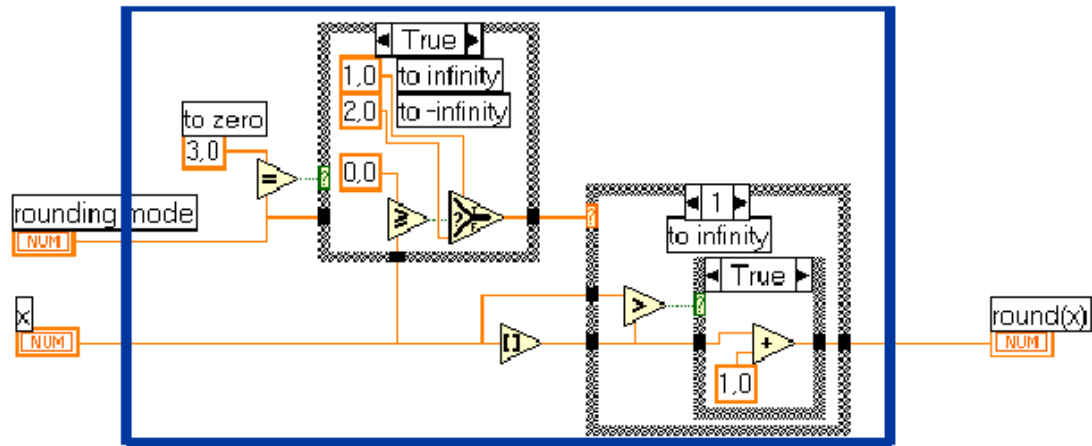


Abbildung 4-9 Die Funktion "floor", Schnappschuss Nr. 2

(neue) Funktionen: greater_equal(), select(), greater(), add().

Verwendung von "floor" in einem neuen Programm: (Icon in blau markiert): SW-Pfad: examples/apps/tempsys.llb/remainder.vi

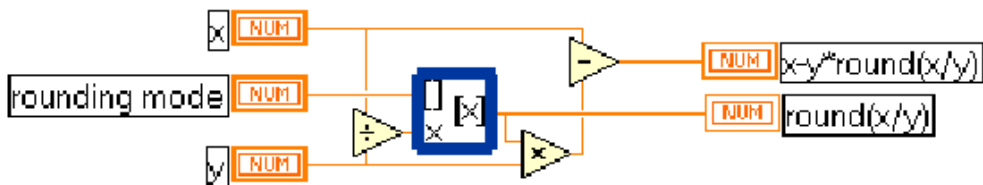


Abbildung 4-10 Einbettung von "floor" in "remainder"

(neue) Funktionen: divide(), multiply(), minus()

User Interface:

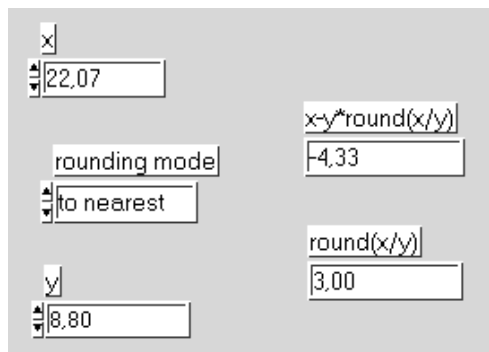


Abbildung 4-11 User Interface von "remainder"

Beispiel 4-2 "incremental filter" mit einer Formel Node

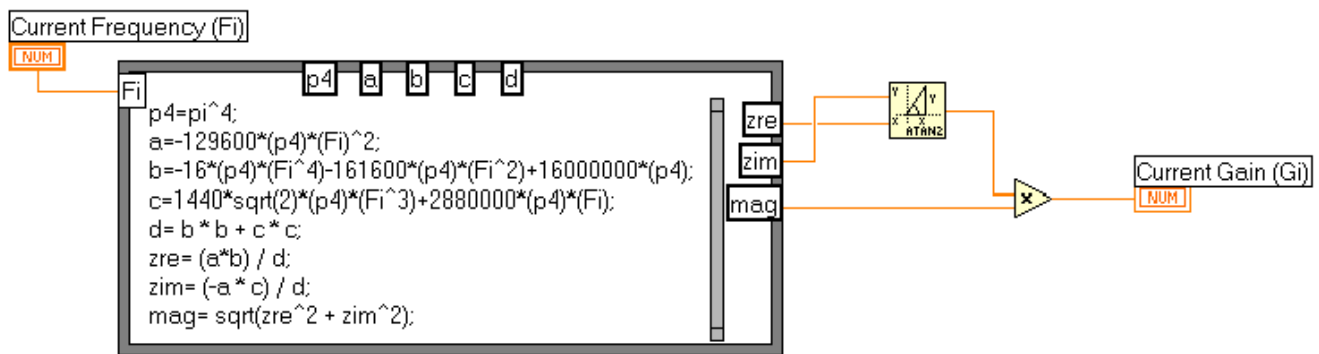


Abbildung 4-12 Eine Formel-Funktion

(neue) Funktionen: formula_node(), atan2(). SW-Pfad: examples/apps/freqresp.llb/incremental filter.vi

User Interface:

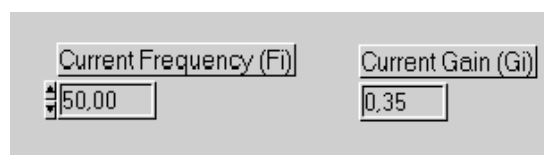


Abbildung 4-13 Oberfläche von "incremental filter"

Bewertung

- keine Schleifen im Datenfluss (nur mit Hilfe der while-Konstrukte)
- keine expliziten Kanäle
- keine zufällige Auswahl (kein Wettbewerb um einen Kanal)
- (auch graphisch) differenzierte Datenstrukturen
- Bündelung von Daten zu Cluster (Records)
- Hierarchiebildung stark ausgeprägt

[Anfang](#)

4.2 Middleware

Das Gebiet "Verteilte Systeme" kann hier natürlich nicht umfassend behandelt werden. Wir beschränken uns auf einige spannende Entwicklungen.

[Anfang](#)

4.2.1 Client/Server Architekturen

Heute bestehen in der kommerziellen DV vier Client/Server-Paradigmen [Or95]:

- SQL-Datenbanken
- TP Monitore
- Groupware und
- Verteilte Objekte.

Die Kommunikationsbeziehungen bei Client/Server Systemen sind nicht immer nur Frage/ Antwort-Paare [DeJ96]:

		Example
Datagram	Single outbound message	Network device status notification
One-shot	Single outbound message; single response	Network- based calculation server
Query	Single outbound message; chained responses	LAN-based database query
Asymmetric	Multiple outbound messages and responses; single session	Customer-service call center
Symmetric	Multiple outbound messages and responses; in two dedicated channels	High-throughput reservation system

Tabelle 1-2 Communication Models

[Anfang](#)

4.2.2 Middleware

Die Entwicklung ist gekennzeichnet durch einen Trend zu komfortablen Software-Schnittstellen, die immer wiederkehrende Aufgaben bei der Kommunikation in Verteilten Systemen übernehmen. Die Programmierung auf der Basis von **Sockets** und **Remote Procedure Calls** wird damit verdrängt. Ein wichtiger Ansatz ist **DCE** (Distributed Computing Environment).

Beispielsweise werden verschiedene "Broker" angeboten, die die Skalierbarkeit vom VS unterstützen, indem der Broker sich freie Ressourcen für anstehende Dienstleistungen selber sucht, z.B. **Common Object Request Broker Architecture (CORBA) 2.0** der Object Management Group's (OMG) [Or95], [Jo96], [Sie96], [Sie96a]. Weiterhin ist **OLE Controls** von Microsoft [Jo96] und der **Task Broker** von HP [HP89] zu nennen. **ToolTalk** der Fa. Sun stellt eine weitere Schnittstelle bereit [SUN??], [Wa96].

Exkurs: Quelle [DeJ96]

Middleware is software that allows **elements of applications** to **interoperate** across network links, **despite differences** in underlying communications protocols, system architectures, operation systems, databases, and other application services.

1. First, middleware isn't meant specifically to link physical clients and servers (that's the job of connectivity protocols such as TCP/IP). Rather, middleware seeks to link the logical elements within applications.
2. Second, the paramount goal of middleware is any-to-any interoperability. Application modules interoperate using a variety of methods, including file exchanges and sharing, shared databases, transactions, and remote procedure calls (RPCs).
3. Third, middleware shields applications from diversity in the underlying environment. Middleware helps your applications operate happily in any environment.

Middleware achieves its purposes by providing

- application-level protocols and formats,
- access to application services,
- support for one or more application models, and administrative facilities.

Middleware products address all three phases of applications development: development, execution, and deployment (including management).

Over the last several years, middleware has expanded in this area. In the early days, middleware was almost totally communications software. Tabelle 4-1 Middleware Components

Development tools	Execution environment	Deployment facilities

<p>Language Bindings</p> <p><i>IDL, SQL, etc.</i></p>	<p>Application Services</p> <p><i>Transaction Management, SQL optimization,</i></p> <p>Core Services</p> <p><i>Name Management, Security,</i></p> <p>Communication Services</p> <p>API Control Service</p> <p>Protocol Service</p> <p>Protocols and Formats</p>	<p>Administrative Tools</p> <p><i>Configuration</i></p> <p><i>Console</i></p> <p><i>Performance</i></p> <p><i>SNMP</i></p>
--	---	---

Of these three components, the execution environment is typically the primary thrust of middleware. Deployment facilities and development tools are often either rudimentary or absent.

[Anfang](#)

4.2.3 Componentware

Eine spannende Diskussion über Bausteine findet sich in [Jo96]:

Enter components - prefabricated parts of **applications that a developer can string together** to create a sum greater than its parts. Prefabricated application building blocks are coming. The only question is when.

The **National Institute of Standards & Technology NIST** (Gaithersburg, MD) administers \$150 million in government funding to find new and better ways to create component-based software. So far, NIST has chosen to fund 16 projects.

At the heart of NIST's program is the goal to make components and the skeleton applications that they plug into easier to build. The benefits to corporate developers would be shorter development time, more-reliable code, and more efficient use of development resources. The program also seeks ways to create a viable component market that spawns an industry of commercial component developers who can expect sales volumes large enough to recoup their programming costs.

In the past, we've had some success by focusing on interface standards, but that's not a long-term solution," "Interface standards have life expectancies shorter than the standards-setting process."

New automation tools would generate both the needed component and the mechanism for gluing it together with other components. If languages and tools address the details of building component-based applications, software developers can focus solely on an application's features, performance, and reliability.

But that desire isn't new. It's what spawned

- CASE,
- object-oriented programming, and
- today's more down-to-earth version of componentware.

Choose one from column A, one from column B, and one from column C. Presto: You have an application. Why shouldn't it be that easy? After all, you can build an entire personal computer by selecting off-the-shelf components. Today's three main component technologies are CORBA, OLE, and OpenDoc.

[Anfang](#)

4.2.4 Kommerzielle Vorschläge

4.2.4.1 ToolTalk

ToolTalk ist ein Nachrichtenvermittlungsdienst, entwickelt von der Fa. Sun [Wa96], [Sun??]. Es ermöglicht die Prozeßkommunikation über Rechner und Plattformgrenzen hinweg mit dem Konzept der Nachrichtenvermittlung (Message Passing). Mit ToolTalk können unabhängige Anwendungen netzwerktransparent kommunizieren.

- **Netzwerktransparent** heißt, daß die sendende Anwendung (der Sender) nicht wissen muß auf welchem Rechner die empfangende Anwendung (der Empfänger) läuft.
- Mit **Kommunizieren** ist gemeint, der Sender erzeugt eine ToolTalk Nachricht, setzt die Attribute und Argumente und sendet sie. ToolTalk ermittelt anhand der In der Nachricht eingestellten Attribute den oder die geeigneten Empfänger und stellt die Nachricht zu.
- **Unabhängig** heißt, der Empfänger erklärt sein Interesse an bestimmten Nachrichten in Form von Nachrichtenmustern. ToolTalk ermittelt durch vergleichen dieser Muster mit jeder Nachricht (Pattern Matching) den oder die Empfänger.

Beispielsweise kann die Anwendung A Nachrichten an Anwendung C versendet und ihrerseits Nachrichten von Anwendung B empfangen. Dabei ist es egal auf welchem Rechner die Anwendungen laufen, sie müssen nur mit der selben ToolTalk Sitzung verbunden sein (Session-Scoping), oder ein gemeinsames Interesse an eine Datei haben (File Scoping).

Damit Anwendungen Nachrichten austauschen können, müssen sie sich auf ein gemeinsames Nachrichtenprotokoll einigen. Ein Nachrichtenprotokoll legt fest, welche Aufgaben eine Anwendung erfüllen kann, d.h. auf welche Nachrichten sie überhaupt reagiert und auf welche Art und Weise sie das macht.

[Anfang](#)

4.2.4.2 Corba 2.0

The **four key elements** of the OMG's architecture are: [Or95]

- **The ORB (Object Request Broker):** It's the object interconnection bus. Clients are insulated from the mechanisms used to communicate with, activate, or store server objects. CORBA 1.1, introduced in 1991, defined the IDL (interface definition language) and APIs that enable client/server object interaction within a specific implementation of an ORB. CORBA 2.0 specifies how ORBs from different vendors can interoperate.
- **Object services:** Packaged as components with IDL-specified interfaces, these services extend the capabilities of the ORB. OMG has adopted the following object services: naming, event notification, persistence, life-cycle management, transactions, concurrency control, relationships, and externalization. Five new services are expected by mid-1995: query, licensing, properties, security, and time. These object services should be bundled with every ORB.
- **Common facilities:** These collections of IDL-defined components define the rules of engagement for application objects. They're categorized as horizontal and vertical. The horizontal ones address four disciplines: user interface, information management, systems management, and task management. The user-interface services, like OLE and OpenDoc, govern on-screen activities such as in-place editing. The information-management services resemble the OLE and OpenDoc mechanisms for compound document storage and data interchange. Systems management services define interfaces used to manage, instrument, install, configure, operate, and repair distributed objects. Task management services include things like work flow, long transactions, agents, scripting, and rules. In the realm of vertical facilities, IDL-defined interfaces will support suites of interacting objects specialized for health, retail, finance, and other domains.
- **Application objects:** These are components specific to end-user applications. To participate in ORB-mediated exchanges, they too must be defined using IDL. Application objects, of course, build on top of services provided by the ORB, common facilities, and object services.

The application objects and common facilities, however, are works in progress. When the common facilities are ready, CORBA will provide IDL interfaces for virtually every distributed service we know today. Note that it is not the OMG's goal to reinvent all of these services, but rather in many cases to provide **IDL wrappers** for existing standards. The ambitious goal of CORBA is to turn everything into nails, and give everyone a hammer. The nails are the IDLized services, and the hammer is the IDL interface to these services.

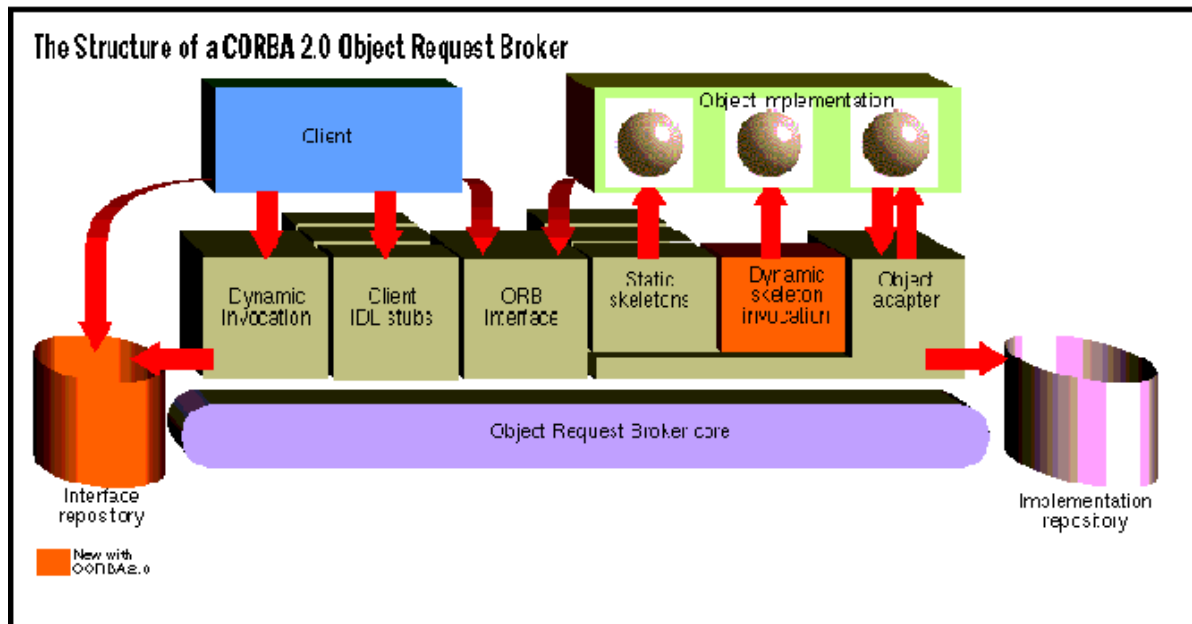


Abbildung 4-14 Structure of a CORBA 2.0 ORB

CORBA clients invoke services statically by way of compiled IDL (interface definition language) stubs or on the fly using of dynamic invocation APIs. CORBA servers route client requests to object implementations analogously by means of static skeletons or at run time using the new dynamic skeleton interface. Clients and servers share the utility APIs of the ORB (object request broker) interface.

[Anfang](#)

4.2.4.3 OLE

OLE2.0 provides the event-handling, file-managing, and information-sharing mechanisms that serve as the plumbing of application components like OLE Controls. Moreover, OLE is an integral part of Windows 95 and other Microsoft operating systems. OLE-based components don't offer inheritance -- the capability to move data and functions from existing objects into new objects for use inside an application.

They transform user-generated events (such as mouse clicks) into messages that communicate with the application (the **container** in OLE parlance). OLE Controls use these events to trigger event handlers that carry out the bidding of the OLE Control.

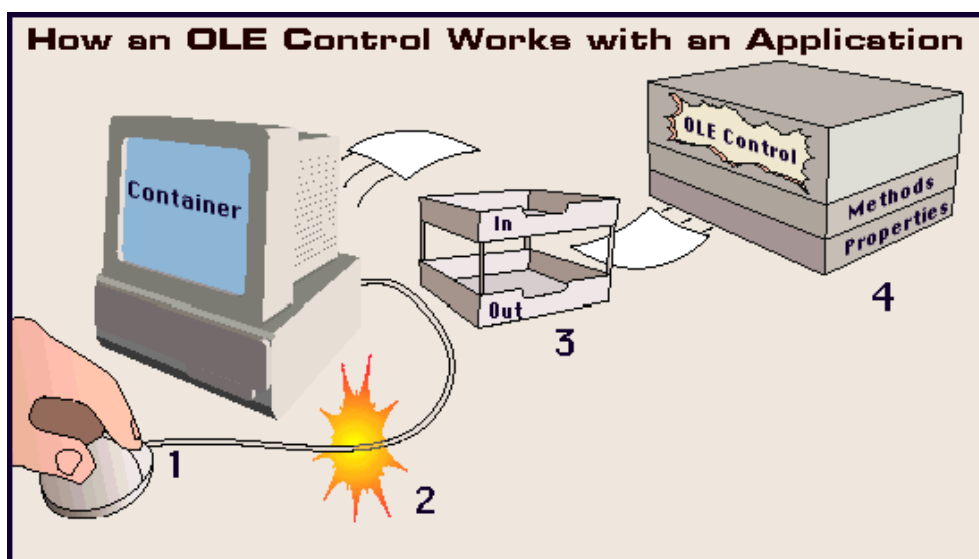


Abbildung 4-15 How an OLE Control Works with an Application

1. User-generated **events**, such as a mouse click, send off a message to the OLE application.
2. The message tells the application (known in OLE parlance as a **container**) what action it should perform.
3. Containers and **OLE controls** communicate through a **messaging interface**. Every OLE control has a built-in receptor, called a **sink**, to receive instructions from an application.
4. Sometimes a user generates an event within an OLE Control. To communicate this event to the container, the control tells the container to dynamically set up a sink to receive the communication.

Sometimes a user generates an event within an OLE Control. To communicate this event to the container, the control tells the container to dynamically set up a sink to receive the communication.

[Anfang](#)

4.2.4.4 OpenDoc

Quelle: [Jo96] **OpenDoc** programs consist of documents, parts, container applications, part editors, part services, and a part viewer. You build applications by grouping OpenDoc parts inside a **document**:

1. **Parts** typically have part editors or part services, which enable users to view or manipulate parts, depending on the application.
2. The **container** is a stand-alone application that a developer has modified to support embedded OpenDoc part editors and services
3. **Part editors** display and modify the contents of a part, as well as provide a user interface for making these modifications. The user interface could include menus, controls, tool palettes, or other elements for interacting with the contents. For example, an OpenDoc part could be a bar graph that will go into an annual report. By invoking a part editor, a user can display and alter the information.
4. **Part services** provide the background features of an OpenDoc part and the user interface for manipulating the contents of that part. For example, a database-access part would build in the database-access functionality as a part service.
5. **Part viewers** allow users to see and print the contents of a part. A viewer is useDul if a developer would like the user to be able to examine but not change the information in an OpenDoc part.

Developers will find OpenDoc a radical departure from traditional GUI application development because it gives them a component discipline that other architectures, including OLE, have yet to match.

- OpenDoc is a set of shared class libraries with a platform-independent interface defined by an **interface definition language (IDL)**.
- It uses object skeletons based on **CORBA-compliant System Object Model (SOM)** base classes.
- It's easy to add new parts at any time because the SOM objects dynamically bind.

The IDL and SOM base of OpenDoc allow part editors built with various compilers and programming languages to talk with each other using a common communications mechanism. SOM includes a component-packaging mechanism. When building OpenDoc objects, developers can use this mechanism to package part editors using binary class libraries for shipment as DLLs; you can send a part to someone and be sure the recipient will have the appropriate viewer to see the part. SOM is the OO heart of OpenDoc, and it's where OpenDoc gets its inheritance capabilities. Inheritance lets developers subclass OpenDoc parts and either use or override methods and data delivered using the DLL binaries. This feature brings the concept of **extendable parts** to OpenDoc, which is one of the real selling points of this technology

Copyright ©; Godbersen, 1996 [Anfang](#) 29.09.96 4.htm

5 Verteilte Funktionsnetze

Inhalt:

5.1 [Anforderungen](#)

5.2 [Lösungen mit Funktionsnetzen](#)

5.2.1 [Eingliederung \(Wrapper\)](#)

5.2.2 [Einbinden von bewährter \(legacy\) Software](#)

5.2.3 [Nebenläufigkeit](#)

5.2.4 [Standort-Transparenz](#)

5.2.5 [Skalierbarkeit](#)

5.3 [Realisierungskonzepte](#)

5.3.1 [Bisherige Lösungen](#)

5.3.2 [Neuer Ansatz](#)

5.4 [Ausblick](#)

[zurück zum Gesamt-Inhaltsverzeichnis](#)

Mit zunehmender Verbreitung von Rechnernetzen wird die Entwicklung Verteilter Systeme immer wichtiger. Einen Ansatz zur Fortschreibung des Funktionsnetz-Konzepts wird gesucht, um die Komplexität dieser Aufgabenstellung zu reduzieren und dabei gleichzeitig den Erstellungsprozeß zu flexibilisieren und effizienter zu gestalten.

Folgende Stichpunkte treten hervor:

- Beherrschung der Komplexität
- Legacy -Software
- Einbettung in Verteilte Systeme
- geeignete Unterstützung des Produktionsprozesses (Workbench)
- Kommunikationsarchitektur für die Workbench

[Anfang](#)

5.1 Anforderungen

Es werden in Stichpunkten allgemeine Anforderungen an Verteilte Systeme formuliert:

1) Mehrfachnutzung von Betriebsmitteln (resource sharing):

- **ökonomische Verwertung von knappen Betriebsmitteln:** Hardware: Drucker, Prozessoren, ..
Software: gemeinsame Lizenz, nur n Nutzer gleichzeitig
- **Bereithaltung von Daten ohne örtliche Begrenzung:** Datenbanken (Geld, Warenlisten, ..), Dateien (Wetterkarte), ..
- **Beibehaltung von Autonomie:** Organisationen können den Zugriff zu ihrem Besitzstand von Betriebsmitteln steuern.

genereller Lösungsansatz: **Client/Server-Modell:**

1. Betriebsmittel (resources) werden nur von Servern gehalten.
 2. Server bieten Dienste (services) an.
 3. Client-Prozesse fordern zur Durchführung ihrer Aufgaben solche Dienste an.
 4. Server bearbeiten Dienstanforderungen auf Anfrage.
 5. Der Austausch der Informationen geschieht über Botschaften. Dazu wird ein Kommunikationsnetz eingesetzt..
-

2) Offenheit (openness):

- Schnittstellen öffentlich publiziert
 - einheitliche Interprozeß-Kommunikation
 - unterschiedliche HW/SW-Anbieter
 - neue Dienste einfach integrierbar (SW-Konzepte)
-

3) Nebenläufigkeit (concurrency):

- skalierbare Leistung durch nebenläufigen Einsatz
 - vieler Benutzer (Client-Prozesse)
 - vieler Server (Rechner , CPUs)
-

4) Skalierbarkeit (scalability):

in unterschiedlichen Größenordnungen einsetzbar

im laufenden Betrieb erweiterbar

Betriebsmittel bei Engpässen ggf. mehrfach bereithalten (reproduzieren)

5) Fehlertoleranz (fault tolerance):

- hohe Verfügbarkeit durch
- Hardware Redundanz
- Recovery- Mechanismen (SW)
- robustes Kommunikationssystem

6) Transparenz (transparency):

- System wird als Einheit gesehen und nicht als eine Sammlung einzelner Komponenten.
- Der aktuelle Standort eines Betriebsmittels wird unwichtig (*gilt für Benutzer und Entwickler*).

Teilaspekte von Verteilungstransparenz sind (vgl. [ISO 10746-1] ODP):

1. **Zugriffstransparenz (access):** Lokale und verteilte Informationsobjekte sind mit derselben Methode erreichbar: *z.B. mail, aber nicht login/rlogin*
2. **Standorttransparenz (location):** Zugriff möglich ohne Wissen, wo sich ein Informationsobjekt befindet.
3. **Nebenläufigkeitstransparenz (concurrency):** nebenläufiger Gebrauch von Informationsobjekten ohne gegenseitige Beeinflussung.
4. **Reproduktionsstransparenz (replication):** Kein Wissen über den Einsatz von Mehrfachinstanzen eines Informationsobjekts (um Verlässlichkeit und Performace zu erhöhen).
5. **Fehlverhaltenstransparenz (failure):** Teil-Systemausfälle werden dem Benutzer nicht sichtbar.
6. **Verlagerungstransparenz (migration):** Verlagerung von Informationsobjekten hat keinen Einfluß.
7. **Leistungstransparenz (performace):** Systemrekonfigurationen zur Leistungsverbesserung erfordern keine Änderungen seitens der Benutzer (*z.B. auch keine Verschlechterung von Antwortzeiten*).
8. **Skalierungstransparenz (scaling):** Variation der System-Größenordnung ohne Auswirkung auf die Systemstruktur oder Anwendungen.

[Anfang](#)

5.2 Lösungen mit Funktionsnetzen

Idee der Funktionsnetze als Entwicklungsumgebung ist das Verbergen von Systemabhängigkeiten und Kommunikationsspezifika.

Viele der oben aufgestellten Forderungen werden durch Funktionsnetze erfüllt: Durch die einheitliche, konsistente Schnittstelle wird insbesondere die Offenheit, Nebenläufigkeit und Skalierbarkeit unterstützt.

[Anfang](#)

5.2.1 Eingliederung (Wrapper)

Der Benutzer soll Bausteine und eine Spezifikation über deren Zusammengehörigkeit im System zur

Verfügung stellen; die Realisierung der Ausführung wird von der Funktionsnetzumgebung übernommen. Dafür ist es notwendig, daß die Bausteine (Funktionen) eine genormte Schnittstelle besitzen.

Um diese Schnittstelle wird ein Kommunikationsrahmen (wrapper) gelegt, der Verbindungen zu allen Kanälen unterhält, mit denen die Instanz der Spezifikation entsprechend verbunden ist. In diesem Rahmen ist der Markenspielalgorithmus der Petrinetze implementiert, also das Vorgehen, das nötig ist, um einen Baustein zu aktivieren.

Dieses Einpacken (wrapper) läßt sich auf unterschiedliche Quellen anwenden:

1. Vordefinierte Instanz mit **Interpreter**, der einen begrenzten Sprachvorrat abarbeiten kann. Keine Compilation nötig. Interpreter arbeitet lokal zur Laufzeit. Die jeweilige aktuelle Ausprägung des Baustein bekommt den spezifischen Algorithmus als Parameter übergeben.
2. Einbinden eines als **Quellprogramm** zur Verfügung stehenden Algorithmus in den Rahmen. Compilierung nötig. Es entsteht ein neuer Baustein.
3. Einbinden eines als nur als **ausführbares Programms** zur Verfügung stehenden Algorithmus in den Rahmen. Die Verbindung zum Rahmen muß dann über eine geeignete Trennung in mehrere Prozesse und Interprozeßkommunikation (z.B. pipes) erfolgen. Compilierung (nur der Rahmens) nötig. Es entsteht ein neuer Baustein.

Das folgende Beispiel zeigt die Eingliederung eines Algorithmus, indem mit Hilfe eines Rahmens nach außen hin eine Instanz erzeugt wird.

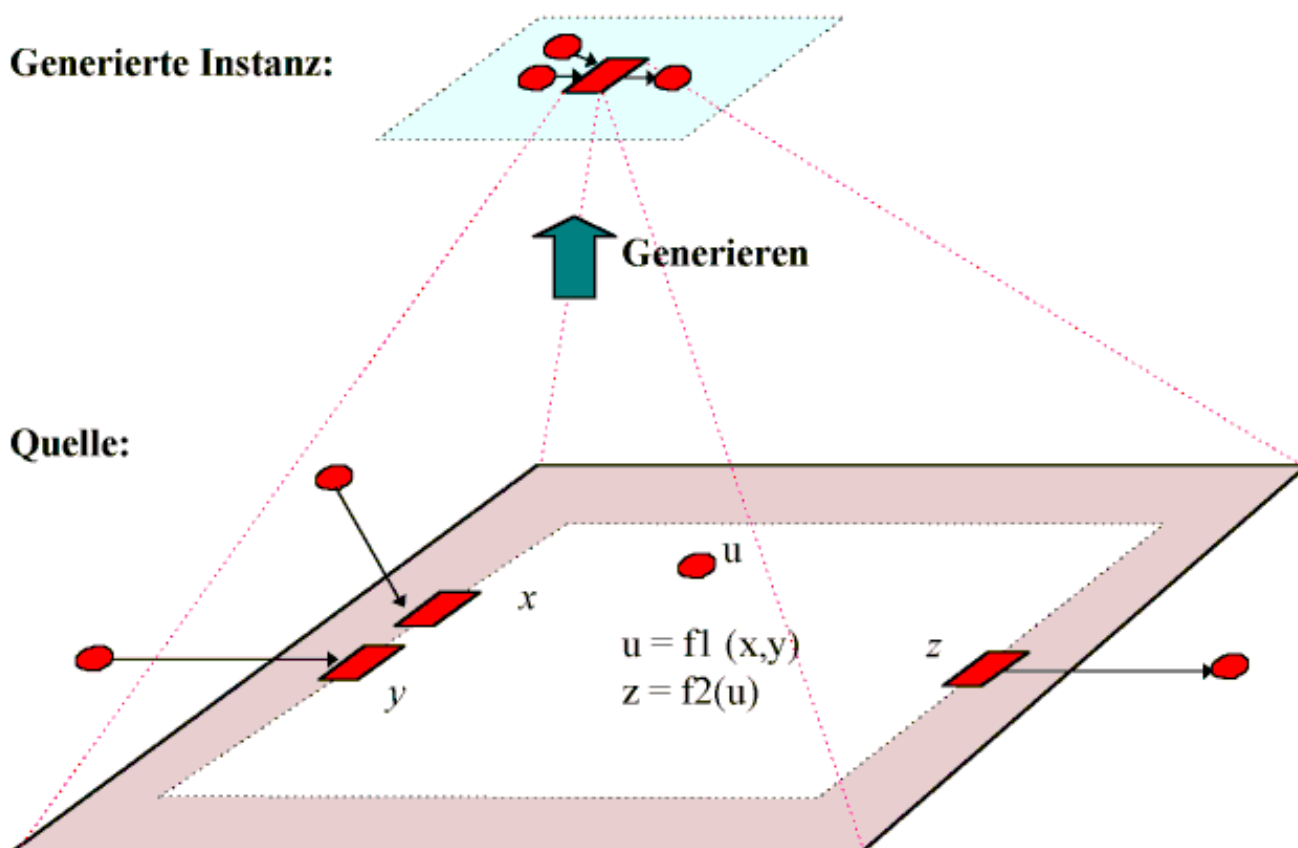


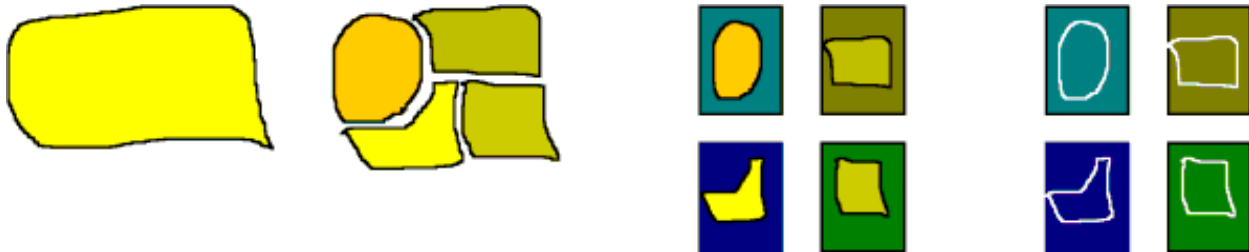
Abbildung 5-1 Ein Algorithmus wird in eine Instanz eingegliedert

Hinweis: Das Vorgehen ähnelt dem **Virtual Manufacturing Device (VMD)** in MAP/MMS Manufacturing Message Standard [ISO 9506].

[Anfang](#)

5.2.2 Einbinden von bewährter (legacy) Software

Die folgenden Abbildung faßt das Vorgehen bei der Einbindung von altbewährter Legacy-Software zusammen:



Legacy-System Modularisierung Eingliederung Archivierung

Abbildung 5-2 Vorgehen bei der Einbindung von Legacy-Software

Nach erfolgter Archivierung können die Komponenten wieder eingesetzt werden, u.a. auch, um genau den alten Leistungsumfang wiederherzustellen. Das Maß an Wiederverwendbarkeit ist natürlich gestiegen.

[Anfang](#)

5.2.3 Nebenläufigkeit

Ein Ziel besteht darin, größtmögliche Nebenläufigkeit zu realisieren. Dies wird als integraler Teil des Modellierungsansatzes implizit bereitgestellt.

[Anfang](#)

5.2.4 Standort-Transparenz

Teilnetze werden zusammengefaßt und einem Rechner zugeordnet. Die Zuordnung ist Teil der Attributierung des Funktionsnetzes und kann jederzeit geändert werden.

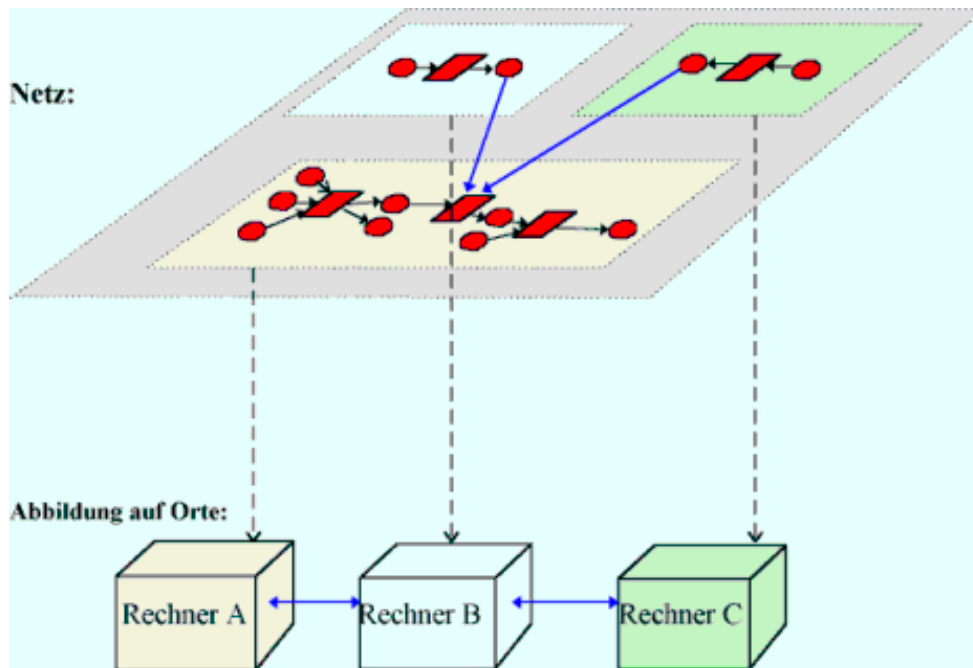


Abbildung 5-3 Standort-Transparenz

[Anfang](#)

5.2.5 Skalierbarkeit

Teilaufgaben können an mehrere Server vergeben werden, indem eine Wettbewerbssituation hergestellt wird:

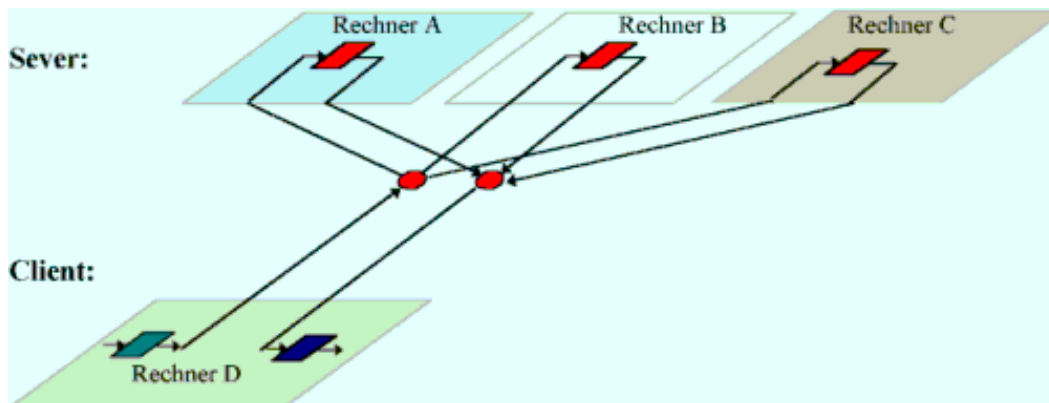


Abbildung 5-4 Nebenläufige Abarbeitung durch multiple Server

[Anfang](#)

5.3 Realisierungskonzepte

Ein bisheriges Ziel bestand darin, größtmögliche Nebenläufigkeit und Unabhängigkeit der Knoten voneinander zu realisieren. Dies muß ergänzt werden durch den ökonomischen Einsatz neuer, mächtiger, dedizierter

Subsysteme.

[Anfang](#)

5.3.1 Bisherige Lösungen

5.3.1.1 Lokale Systeme

In den ersten Jahren wurden lokale Lösungen u.a. auf der Basis von SIMULA realisiert. Das gesamte Programmsystem lief als einzelner Prozeß. Daran schloß sich eine Portierung auf UNIX an, bei der mehrere Instanzprozesse bereitgestellt wurden, die Steuerung des Ablaufs aber durch einen "Hauptprozeß" erfolgte (s. [Jä90]).

[Anfang](#)

5.3.3.2 Autonome Verteilte Systeme

Um eine völlige Autonomie der Teilkomponenten und eine größtmögliche Freiheit bei Wettbewerbssituationen zu gewährleisten, wird jeder terminale Knoten des Modells auf einen (Betriebssystem-) Prozeß abgebildet. Instanzprozesse bestehen dabei aus dem Baustein und einem kommunikationsfähigen Rahmen. Kanalprozesse stellen die Synchronisationspunkte dar. Sie verwalten die Daten, die zwischen den Bausteinen ausgetauscht werden. Durch die feste Zuordnung der statischen Modellinformationen auf Prozesse ist es möglich, ein relativ einfaches Protokoll zwischen den Prozessen zu installieren. Es folgt eine Skizze der Lösung:

Ein Instanzprozeß prüft seine Eingangskanäle, ob sie Marken für ihn bereitstellen können. Diese Konzeption führt dazu, daß Instanzprozesse als Auftraggeber (*lokaler client*) realisiert werden. Sie rufen mit ihren Anfragen Dienste der Kanalprozesse auf. Diese arbeiten als Auftragnehmer (*lokaler server*). Kanalprozesse sind als abstrakte Datentypen implementiert, ihre Operationen (Dienste) können in abfragende und modifizierende Operationen unterteilt werden.

Durch diese Differenzierung in Auftraggeber und Auftragnehmer und die Möglichkeit, den Kanalprozeß als abstrakten Datentypen zu realisieren, bietet sich für die Kommunikation die Benutzung von Prozedurfernaufrufen (*remote procedure calls* RPCs) an.

Kanalprozesse, die die Synchronisation übernehmen, können zu einem Zeitpunkt nur die Anfrage eines einzelnen Instanzprozesses bearbeiten; sie stellen somit einen kritischen Abschnitt dar. Damit sie nicht zu Engpässen in der Kommunikation des Systems werden, enthalten die Dienste einfache Algorithmen. Die gesamte Steuerung des Schaltverhaltens liegt im Instanzprozeß. Dieser versucht, Marken auf allen Eingangskanälen zu reservieren. Er konsumiert sie erst, wenn auf jedem seiner Eingangskanäle eine Marke reserviert werden konnte. Gelingt dies nicht, müssen die bereits reservierten Marken wieder freigegeben werden. Ist auf einem Eingangskanal keine Marke vorhanden, blockiert sich der Instanzprozeß - er "legt sich schlafen". Geweckt wird er durch ein Signal von dem Kanal, wenn auf diesen eine Marke gelegt wird. Die folgende Tabelle gibt eine Übersicht über die Dienste eines Kanals:

Dienst	Aktion
marke_reservieren	eine Instanz des Nachbereiches wird eine Marke reserviert.

marke_freigeben	Eine bereits reservierte Marke wird von einer Instanz freigegeben, wenn diese nicht schalten kann.
marke_konsumieren	reservierte Marke wird vom Kanal entfernt und an eine Instanz geliefert.
marke_produzieren	Instanz legt eine Marke auf einem Ausgangskanal ab.
weckauftrag_geben	Eine Instanz informiert den Kanal, daß sie auf ein Wecksignal wartet.
wecken	Der Kanal sendet ein Wecksignal, wenn er (mindestens) eine Marke von einer Instanz seines Vorbereiches erhält.

Tabelle 5-1 Dienste eines Kanals

Diese Lösung ist in Diplomarbeiten bereits prototypisch erfolgreich realisiert worden (s. [Pf91]).

[Anfang](#)

5.3.2 Neuer Ansatz

Ziel bezüglich der Werkzeugunterstützung bildet die Einbindung kommerzieller Lösungen. Zu diesem Zweck ist der Einsatz von Middleware-Lösungen (Broker, ToolTalk etc.), Componentware und GUI-Generatoren vorgesehen.

Die folgende Abbildung faßt den Ansatz zusammen:

- Eine Client enthält lokale Knoten und die Benutzeroberfläche (GUI).
- Zwei Server stehen zur Verfügung (einer stellt Daten bereit, der andere enthält Componentware).
- Die Koordination wird von der Middleware übernommen.

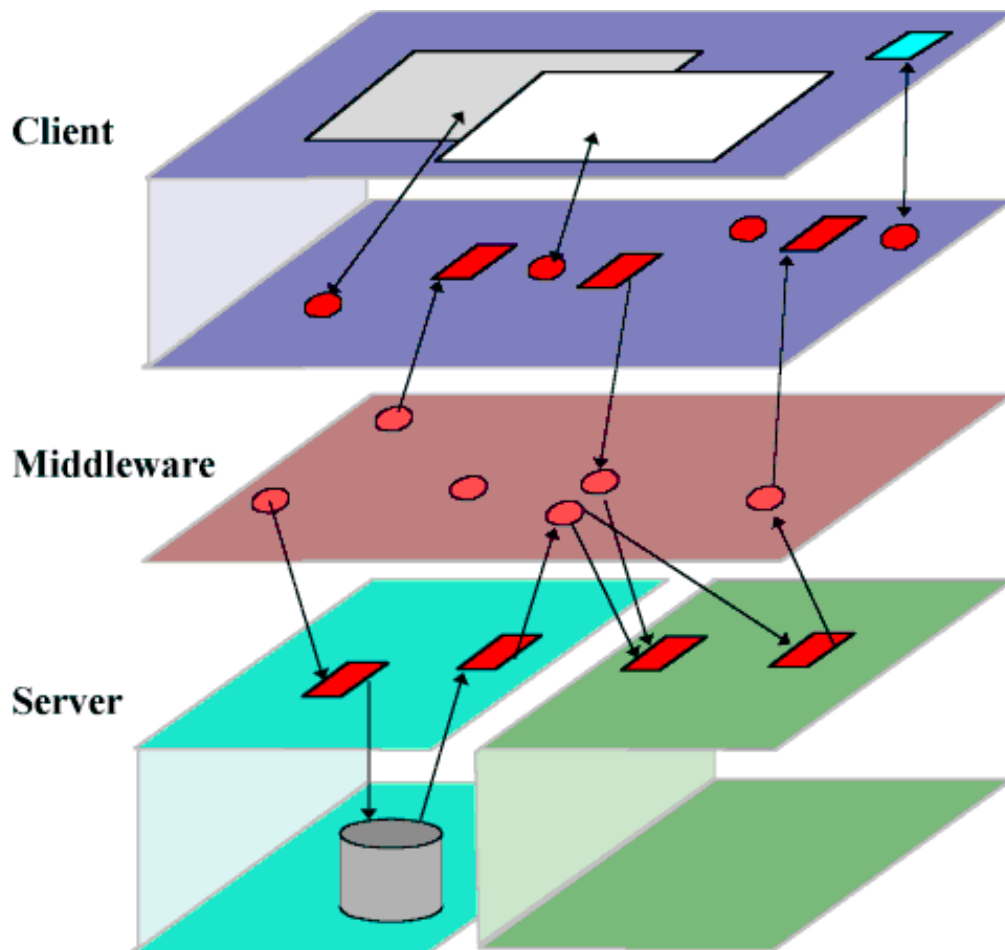


Abbildung 5-5 Funktionsnetze in Verteilten Systemen

Anmerkung: Auf der jeweiligen Ebene sind lokale Kommunikationsbeziehungen (Pfeile) nicht eingezeichnet, um die Übersicht zu erhöhen.

Es wird auf die Prozeß-Abbildung von Kanälen verzichtet. Vielmehr übernehmen die GUI-Umgebung und der Broker einen großen Teil der Aufgabe. Grund: Überwiegend werden Kanäle zum Austausch einzelner Datenobjekte zwischen genau zwei benachbarten Instanzen eingesetzt. Für den Fall von Wettbewerbssituationen kann ja gerade ein Broker eingesetzt werden.

Weiterhin bleibt das Funktionsnetz der Baukasten. Gegenüber den vorherigen Ansätzen muß aber eine gezielte Abbildung in das jeweilige Gastsystem erfolgen. Als Entscheidungskriterium bietet sich eine Erweiterung die ja bisher schon bekannte Zuordnung von Instanzen zu Rechnern auf Rechner-Softwarekomponenten an.

[Anfang](#)

5.4 Ausblick

Verteilte Funktionsnetze können unter Verwendung der im folgenden Kapitel vorgestellten Workbench VerSoS aufgebaut werden.

Die vorgeschlagene Lösung läuft auf eine "Entmachtung" der Kanäle hinaus, die vor allem in den bisherigen Anwendungsgebieten (man denke an die Simulation von Warteschlangen) weitaus präzenter war. Jetzt schlüpfen die Kanäle mehr in die Rolle von Schnittstellen und Knoten zum Starten von Nebenläufigkeit.

Anschlußarbeiten sind u.a. auf folgenden Gebieten vorzunehmen:

1. Verbindung von GUI-Event Handling mit der FN-Schaltregel
2. Anbindung von FN an ToolTalk
3. Untersuchung zur Wahl der Datenstrukturen an den Schnittstellen.

Copyright ©; Godbersen, 1996 [Anfang](#) 29.09.96 5.htm

6. Die Workbench VerSoS

Inhalt:

[6.1 Basis](#)

[6.2 Vorgehensmodell bei der Softwareerstellung](#)

[6.2.1 Benutzerprofile](#)

[6.3 Die Workbench Werkzeuge Datenbestände](#)

[zurück zum Haupt-Inhaltsverzeichnis](#)

Ziel von VerSoS (**V**erteilte **S**oftware **S**ynthese) ist es, eine Umgebung zu schaffen, in der verteilte Anwendungen auf einer graphischen Oberfläche spezifiziert, synthetisiert und unmittelbar ausgeführt werden können. [Go92], [Kr91], [Pf91]

[Anfang](#)

6.1 Basis

Die folgende Abbildung zeigt die Eigenschaften der für VerSoS erweiterten Funktionsnetze in einer Übersicht:

Eigenschaften der Funktionsnetzelemente





Kante		Nachrichtenfluß Kontrollfluß Kopierende Kante
Kanal/Marke		Typbindung 4 Zugriffsarten Individualisiert Kapazität
Instanz		Tätigkeit Partielles Schalten Faltung Ausnahmesituation
Knoten		Hierarchisierbar Mehrfachdarstellung Extern

Abbildung 6.1 Eigenschaften der Netzelemente

Die Kennzeichnung *externer* Knotenelemente ist eingeführt worden, um die Einbindung externer Datenbestände (z.B. Datenbank) und fertiger Software zu beschreiben.

[Anfang](#)

6.2 Vorgehensmodell bei der Softwareerstellung

Die Software-Erstellung auf der Grundlage des phasenübergreifenden Funktionsnetz-Ansatzes bietet eine Reihe von Vorteilen, beispielsweise die intuitiv verständliche und präzise graphische Darstellung von Systemen, eine integrierte Dokumentation, evolutionäres Prototyping durch den Gebrauch operationaler Modelle und die Wiederverwendung von Software-Bausteinen.

1. **Graphische Spezifikation.** Der Einsatz einer graphischen Methode hat vielfältige Vorteile bei der Spezifikation von Software-Systemen: Durch die Verwendung eines begrenzten Symbolvorrats mit festgelegter Bedeutung ist der Entwurf eindeutig. Komplexe Systeme sind leicht überschaubar, weil grobe Zusammenhänge und Strukturen bereits durch oberflächliche Betrachtung erkennbar sind. Verständlichkeit wird dadurch erreicht, daß kein fachspezifisches Wissen zum Verständnis der Darstellung erforderlich ist. Im Gegensatz zu einer textuellen Beschreibung sind Graphiken vor allem bei der Spezifikation nebenläufiger Systeme, wie es verteilte Anwendungen sind genauer. Durch die intuitiv verständliche und präzise graphische Darstellung resultiert eine verbesserte Kommunikation zwischen Entwicklern und Anwendern.
2. **Evolutionäres Prototyping.** Durch den Rückgriff auf eine Bausteinbibliothek ist eine modellierte

Anwendung jederzeit ausführbar. Für Bausteine, die noch nicht in ihrem vollen spezifizierten Umfang bereit stehen, werden "Dummies" eingesetzt (**mock up**). Durch diese Vorgehensweise kann ein evolutionäres Prototyping umgesetzt werden, Zwischenergebnisse werden schnell erreicht. Die durch Einsatz eines Prototypen möglich gewordene frühzeitige Rückkopplung mit dem Auftraggeber hilft bei der **Validation** des Systems. Damit können Mißverständnisse zwischen Entwicklern und Kunden in frühen Phasen des Lebenszyklusses aufgedeckt werden, was zur Minimierung der Entwicklungskosten beiträgt.

3. **Einheitliche Schnittstellen des Werkzeugverbundes.** Unsere methodenbasierte Software-Produktionsumgebung vermeidet Informationsverlust, wie er bei der Übertragung einer Systemspezifikation zwischen Werkzeugen mit unterschiedlichen Schnittstellen entsteht. Die Software-Entwicklung mit Funktionsnetzen entbindet den Entwickler auch von der mühseligen Aufgabe, Spezifikationsergebnisse "von Hand" zu implementieren und jeden Einzelschritt zu verifizieren. Die Verwendung einer einheitlichen Werkzeugschnittstelle begünstigt außerdem - falls erforderlich - die erneute Bearbeitung eines Zwischenergebnisses in früheren Phasen.
4. **Effizienzsteigerung durch Wiederverwenden von Software-Bausteinen.** Die Wiederverwendung von Software ist ein Schlüssel zur Steigerung der Produktivität des Erstellungsprozesses. Software-Wiederverwendung muß jedoch geplant werden, dies beginnt schon bei der Bereitstellung der Bausteine. Bausteine, die für den wiederholten Einsatz vorgesehen sind, verursachen in der Entwicklung etwa doppelt so hohe Kosten wie herkömmliche Moduln. Diese Kosten können sich nur dann amortisieren, wenn die mehrfache Nutzung aktiv unterstützt wird. Sowohl die Entwicklung neuer Bausteine als auch die Einbindung bereits vorhandener Software-Moduln in eine Anwendung werden durch die Integrierte Dokumentation und die einfache Zusammenfügbarkeit der Software erleichtert.
5. **Integrierte Dokumentation.** Die Dokumentation eines Systems ist bei seiner Spezifikation integriert. Durch die werkzeuggestützte Einbindung der Dokumentation in den Entwicklungsprozeß erhöht sich die Chance, eine konsistent dokumentierte Anwendung zu erstellen. Die Beschreibung mittels Integrierter Dokumentation gibt somit immer den aktuellen Stand der Implementierung wieder. Dies ist wie im Falle der einheitlichen Schnittstelle - bei der Überarbeitung des Systems von besonderer Bedeutung.
6. **Nebenläufigkeit im Entwicklungsprozeß.** Schließlich bietet die Trennung von Modellierung des Systems durch den Entwickler und Implementierung der Bausteine durch den Programmierer die Möglichkeit zur nebenläufigen Bearbeitung (s. Abbildung). Voraussetzung für die Implementierung ist die Spezifikation eines Bausteines in der Integrierten Dokumentation. Umgekehrt kann ein bereits bestehender Baustein in Modellen eingesetzt werden, da sein Eintrag dem Modellierer die zugehörige Dokumentation zur Verfügung stellt.

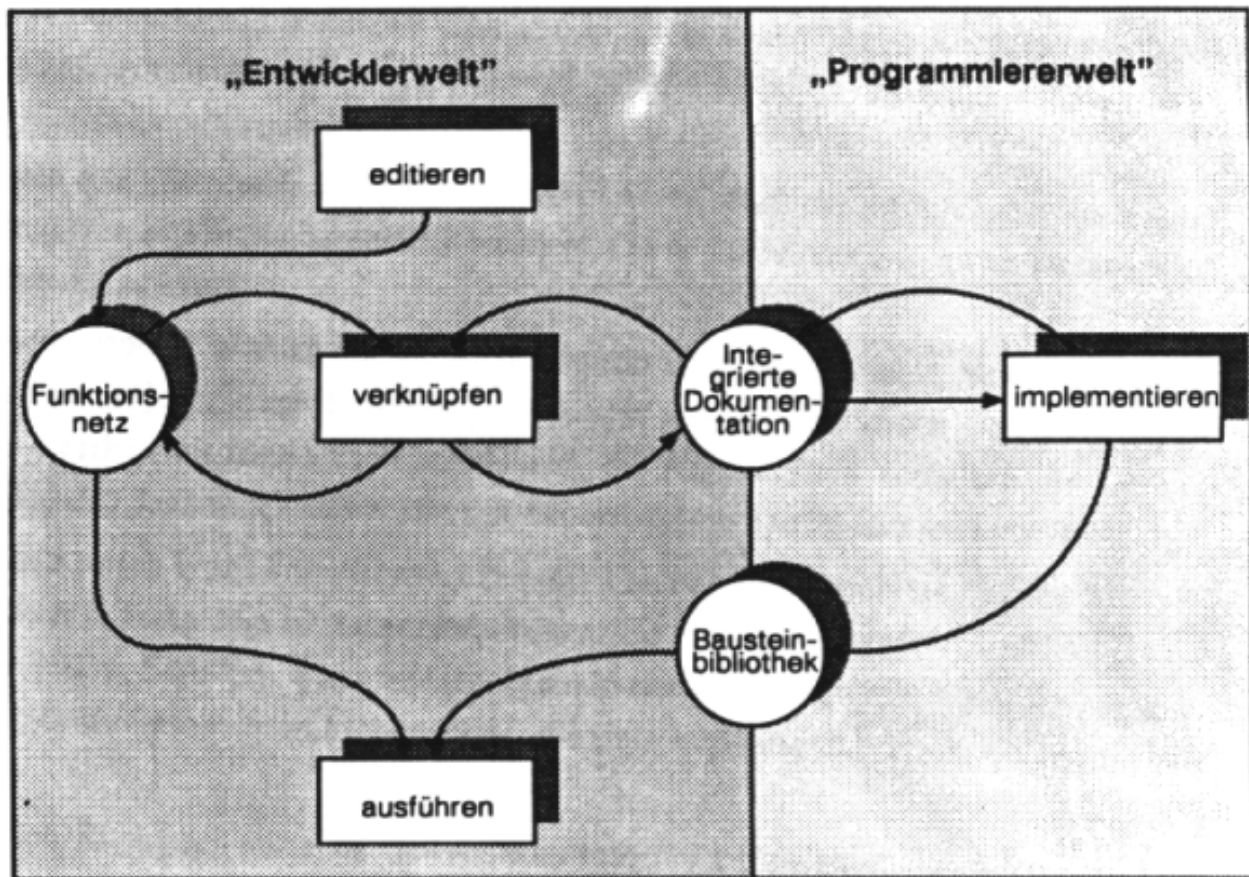


Abbildung 6.2 Nebenläufigkeit im Entwicklungsprozeß

Als weitere (abschließende) Operation sei noch die **Generierung** von "Stand-Alone-Lösungen" genannt.

Der Notwendigkeit, "maßgeschneiderte Standardsoftware" zu produzieren, wird damit Rechnung getragen.

[Anfang](#)

6.2.1 Benutzerprofile

Es können vier Personengruppen unterschieden werden:

1. **Entwickler** der verteilten Anwendungen: Die Aufgabe des Entwicklers ist die Erstellung verteilter Anwendungen nach den Vorgaben eines Kunden. Dazu ist die Bedienung eines Systems mit graphischer Oberfläche und der Umgang mit einer Sprache der 4. Generation (SQL) nötig. Kenntnisse der in der Bibliothek vorhandenen Bausteine sind von Vorteil. Wenn ein benötigter Baustein noch nicht in der Bibliothek vorhanden ist, muß der Entwickler dessen Funktionalität im Lexikon spezifizieren
2. **Programmierer** der Bausteine: Diese Aufgabe beinhaltet die Erstellung und Verwaltung der Software-Bausteine. Die Beherrschung der üblichen Werkzeuge zur Implementierung kleinerer Software-Moduln, wie z. B. Editor, Übersetzer, Debugger, Software-Bibliotheksverwaltung, IINIX-Werkzeuge, etc. wird vorausgesetzt. Darüber hinaus kann die Benutzung von Programmgeneratoren (z. B. ISODE 7) zur Erstellung von Anwendungen in verteilten Systemen nötig sein. Weiterhin soll der Programmierer mit der

Benutzung relationaler Datenbanken vertraut sein.

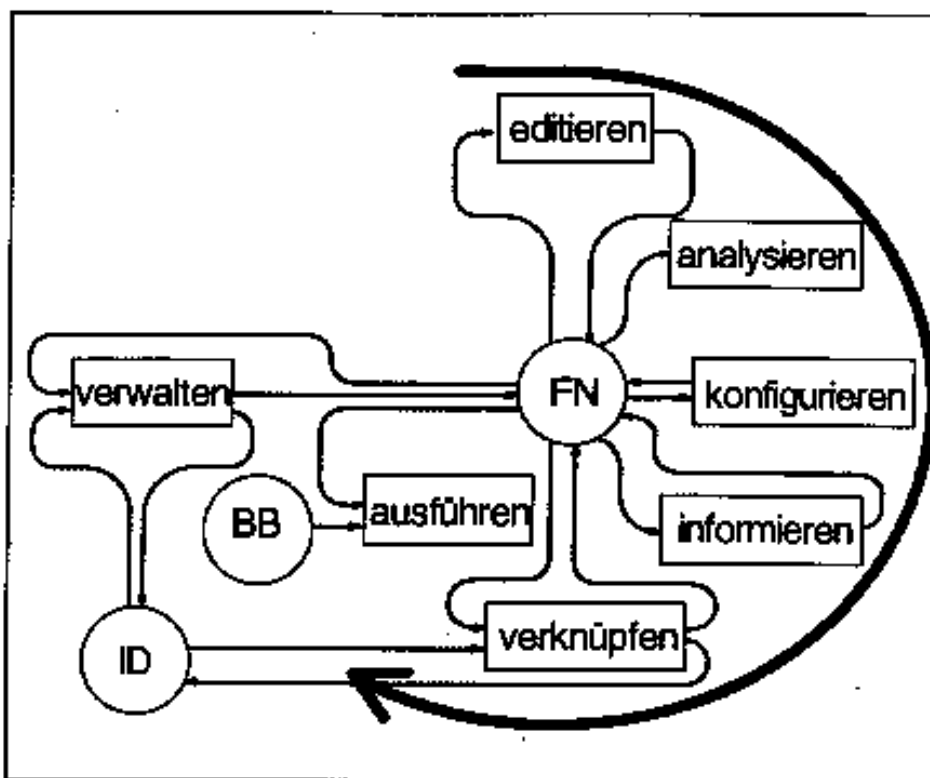
3. **Administrator** ist mit der Pflege der verteilten Anwendung während des Einsatzes betraut. Seine Aufgaben sind die Installation der verteilten Anwendung und der Austausch einzelner Elemente im Wartungsfall. Die dazu notwendigen Arbeiten sind im wesentlichen das Editieren auf der graphischen Oberfläche und die Verknüpfung von Knotenelementen mit Bausteinen. Die vorauszusetzenden Fähigkeiten des Administrators sind in etwa die eines geübten PC-Benutzers. Wenn - etwa in einem kleineren Betrieb - keiner der Beschäftigten diese Funktion ausüben kann, wird sie vom Entwickler übernommen.
4. **Nutzer** der verteilten Anwendung ist typischerweise ein Sachbearbeiter oder Ingenieur. Vom ihm werden keinerlei spezielle Computer-Kenntnisse erwartet. Allerdings sollte er mit der Bedienung von graphischen Benutzungsoberflächen vertraut sein.

Die Personen der ersten beiden Gruppen sind Angehörige eines Software-Hauses, die der letzten beiden sind bei einem Endanwender beschäftigt.

[Anfang](#)

6.3 Die Workbench

Verschiedene Werkzeuge, die den **gesamten Lebenszyklus** einer verteilten Anwendung instrumentell unterstützen, sind über die Darstellung als Funktionsnetz verbunden. **Werkzeuge** repräsentieren Tätigkeiten und werden deshalb bei der folgenden Darstellung als Netz durch Instanzen (eckige Knotenelemente) symbolisiert. **Datenbestände** werden als Kanäle (runde Knoten) abgebildet. Die folgende Abbildung zeigt die Workbench im Überblick. Einzelne Knoten können verfeinert sein.



ID = Integrierte Dokumentation
BB = Bausteinbibliothek
FN = Funktionsnetz

Abbildung 6.3 Systemmodell der Workbench und typischer Bearbeitungszyklus

Im folgenden werden die einzelnen Werkzeuge und Datenbestände näher erläutert:

[Anfang](#)

6.3.1 Werkzeuge

Die Spezifikation einer Anwendung erfolgt durch das **Editieren** eines Systemmodells in der Beschreibungssprache der Funktionsnetze auf einer graphischen Oberfläche. Die Initialisierung eines Netzes, also die anfängliche Verteilung von Marken auf Kanäle, wird **Informierung** genannt. Ein Vorteil beim Einsatz einer Petrinetz-basierten Methode liegt in der Zugänglichkeit der erstellten Modelle zur **Analyse**, die aus einem statischen Teil und einem dynamischen Teil besteht. Verteilte Anwendungen werden in einer heterogenen Rechnerumgebung ausgeführt. Die **Konfiguration** dieser Umgebung muß vom Entwickler angegeben werden. Ein Systemmodell wird zu einem ausführbaren Programmsystem, indem seine einzelnen Knoten jeweils mit Bausteinen **verknüpft** werden. Instanzen sind aktive Knoten, sie werden mit Bausteinen verbunden, die Algorithmen enthalten. Ihr Gegenstück sind die passiven Kanäle, die mit Datenstrukturbausteinen verknüpft werden. Sind alle Knoten eines Funktionsnetzes mit Bausteinen verbunden, ist die Systemkonfiguration festgelegt und das Netz informiert, kann es **ausgeführt** werden. Dabei wird bei jedem Schalten einer Instanz (gemäß der Petrinetzschaltregel) ihr zugehöriger Algorithmusbaustein aktiviert und die dort implementierte Aktion somit umgesetzt. Für die Ausführung des Modells werden nun von einer Ausführungskomponente, die dem Anwender verborgen bleibt, die Bausteine aus der Bibliothek extrahiert und gemäß den Konfigurationsvorgaben auf die Rechner verteilt. Die Ausführungskomponente ist außerdem für den Aufbau von Kommunikationsbeziehungen (Datenflüsse zwischen Bausteinen) gemäß der Modellspezifikation verantwortlich. Stand der Implementierung, Spezifikation von Schnittstellen, Versionen und Releases etc. müssen **verwaltet** werden.

[Anfang](#)

6.3.2 Datenbestände

Die Spezifikation einer verteilten Anwendung wird als **Funktionsnetz** abgelegt. Diese Beschreibung ist für alle Werkzeuge in allen Phasen der Software-Synthese verbindlich. Die Dokumentation einer durch ein Funktionsnetz realisierten verteilten Anwendung erfolgt ebenso wie die Dokumentation der sie realisierenden Bausteine - mittels **Integrierter Dokumentation**. Dort werden auch Informationen zur Verwaltung der Software gespeichert. Zur Ausführung der modellierten Anwendung müssen die terminalen Knoten eines Netzes mit Bausteinen aus einer **Bibliothek** verknüpft sein. Diese Bausteine enthalten Algorithmen (in der Verbindung mit Instanzen) bzw. Datenstrukturen (in der Verbindung mit Kanälen). Die Schnittstellen der Bausteine sind in der Integrierten Dokumentation spezifiziert. So können syntaktische Überprüfungen zur Typverträglichkeit von miteinander verbundenen Instanzen und Kanälen bereits zur Editierzeit erfolgen.

Copyright © Godbersen, 1996 [Anfang](#) 29.09.96 6.htm

7 Literatur

Inhalt:

1. [Direkter Bezug zu Funktionsnetzen](#)
 2. [Hintergrundliteratur](#)
-

[zurück zum Haupt-Inhaltsverzeichnis](#)

7.1 Direkter Bezug zu Funktionsnetzen

- [Bau83] Bauerfeld, W.:
Performance Prediction of Computer Network Protocols.
Proc. Int. Conf. On Communication, Boston, Ma. IEEE (June 1993)
- [Bi81] Bierhaus, L.; Osterbuhr, H.; Pättsch, O.:
Entwicklung und Implementierung eines Modell- und Methodenbank-Monitorsystems im Rahmen von BOSS.
Diplomarbeit, TU Berlin, 1981
- [Br81] Bruhn, H.; Graffigna, A.:
Analyse und Simulation von Funktions- und Petrinetzen.
Diplomarbeit, TU Berlin, 1981 (Betreuer: Godbersen)
- [Gi89] Gies, P.:
Entwicklung eines Netzeditors unter X-Window mit XView und SunGKS.
Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen), 1989.
- [Go77] Godbersen, H.P.; Meyer, B. E.:
Function Nets and System Dynamics.
TU Berlin, Interner CIS-Bericht, 12/77
- [Go78] Godbersen, H.P.; Meyer, B. E.:
Function Nets as a Tool for the Simulation of Information Systems.

- Proc. Summer Computer Simulation Conference, AFIPS, Newport Beach, Ca., 1978, p. 46-53
- [Go79] Godbersen, H.P.:
Funktionsnetze. Ein Ansatz zur Beschreibung, Analyse und Simulation soziotechnischer Systeme.
in: Mayr, H.C.; Meyer, B.E.: (ed.): Formale Modelle für Informationssysteme, Informatik Fachbericht Bd. 21, 1979, p. 146-265
- [Go80] Godbersen, H.P.; Meyer, B.E.:
A Net Simulation Language.
Proc. Summer Computer Simulation Conference, AFIPS, Seattle, Wa., 1980, p. 188-193
- [Go81] Godbersen, H.P.:
Modeling and Analysis with Function Nets.
Proc. 2. European Workshop on Application and Theory of Petri Nets, Bad Honnef, 1981
- [Go81a] Godbersen, H.P.:
Mengengerüst und Verteilungsaspekte in Informationssystemen.
TU Berlin, Interner CIS-Bericht, 02/81
- [Go82] Godbersen, H.P.:
On the Problem of Time in Nets.
in: Girault, C.; Reisig, W. (eds.): Application and Theory of Petri Nets. Informatik Fachberichte, Bd. 52, 1982, p. 23-30
- [Go83] Godbersen, H.P.:
Funktionsnetze. Eine Modellierungskonzeption zur Entwurfs- und Entscheidungsunterstützung.
Ladewig Verlag, Birkach, Berlin, München, 1983, ISBN 3-88924-007-0
- [Go83a] Godbersen, H.P.:
Simulation with "FUN".
Angewandte Informatik, 5/83, pp. 213-219
- [Go83b] Godbersen, H.P.:
On Decision Support with Function Nets.
TU Berlin, Interner CIS-Bericht 03/83
- [Go83c] Godbersen, H.P.:
Informationssysteme auf Mikrocomputern.
TU Berlin, Interner CIS-Bericht, 04/83
- [Go84] Godbersen, H.P.:
Prototyping with Prototypes?
in: Budde et al. (eds.): Approaches to Prototyping, LNCS, 1984
- [Go86] Godbersen, H.; Trümner, H.:
Codegenerierung mit Funktionsnetzen.
Interner CIS-Bericht 6/86, TU Berlin, Dez. 1986
- [Go86a] Godbersen, H.P.; Trümner, H.:

- Ein graphisch orientierter Ansatz zur Bildung operationaler Modelle.
Proc. GI-Jahrestagung, Informatik Fachberichte Bd. 126, 1986, p. 344-355
- [Go90] Godbersen, H.P.; Rastgooy, K.; Schwidder, K.:
Generierung von Programmsystemen aus graphischen Spezifikationen.
Proc. 3rd Seminar on Modelling, Evaluation and Optimization of
Dependable Computer Systems , Wendisch Rietz, Nov. 1990, erschienen
in: Informatik informationen repute, Akademie der Wissenschaften der
DDR, Bd. 9/90, pp. 5-22
- [Go92] Godbersen, H.P.; Kroll, A.; Pfafferott, A.:
Netzbasierte Software-Synthese verteilter Anwendungen.
Proc. Softwaretechnik in Automation und Kommunikation (STAK '92),
VDI Berichte Bd. 937, 1992, p. 199-210
- [Go93] Godbersen, H.P.:
Netzbasierte Softwaretechnik in Verteilten Systemen.
erscheint in: TFH-Forschungsberichte, 1993
- [He90] Heldner, R.:
Remote Procedure Calls und verteilte Systeme
Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen) 1990
- [Jä90] Jänicke, D.:
Funktionsnetzsimulator.
Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen) 1990
- [Jo81] Jochum, F.; Winter, D.:
ISAC - Eine Analyse- und Entwurfsmethode für komplexe
Softwaresysteme. GI-Jahrestagung, Informatik Fachberichte Bd.50 (1981)
- [Kr91] Kroll, A.:
PIE. Eine auf Funktionsnetzen aufbauende Software-
Produktionsumgebung zur Erstellung und Ausführung verteilter
Anwendungen.
Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen) 1991
- [Ku79] Kuley, P.; Kroll, J.:
Entwicklung und Implementierung eines Simulationsmodells auf der Basis
von Funktionsnetzen (Instanzennetzen).
Diplomarbeit, TU Berlin, (Betreuer: Godbersen) 1979
- [La95] Landgraf, M.:
Visualisierung und Programmierung mit dem IRIS Explorer
Diplomarbeit, TFH Berlin, SS95, Betreuer: Prof. Godbersen
- [Le84] Leszak, M.:
Erfahrungen bei der Portierung des Software-Werkzeuges "FUN" von VM-
auf das MVS-Betriebssystem.
Interner Bericht, Kernforschungszentrum Karlsruhe, IDT (1984)
- [Le85] Leszak, M.; Godbersen, H.P.:
DAEMON: A Tool for Performance- Availability Evaluation of

- Distributed Systems based on Function Nets.
Proc. ACM-SIGMETRICS/IEEE Intern. Workshop on Timed Petri Nets,
Torino, July, 1985 pp. 152-161
- [Le86] Leszak, M.:
Modellierungs- und Simulationsinstrumentarium für fehlertolerante
verteilte Systeme angewendet auf verteilte Datenbankverwaltungssysteme.
Dissertation, TU Berlin, Institut für Angewandte Informatik, 1986
- [Me81] Meyer, B.E.:
COLAN - A Language for the Communication Between Simulation
Models. Proc. Summer Computer Simulation Conference, Washington, DC
(July 1981), AFIPS Press
- [Pa83] Pallmann, R.; Trümner, H.:
FUN-Benutzerhandbuch.
TU Berlin, Institut für Angewandte Informatik, Interner CIS-Bericht 8/83
- [Pa84] Pallmann, R.; Trümner, H.:
Entwurf und Implementierung eines Programmgenerators auf der Basis
von Funktionsnetzen.
Diplomarbeit, TU Berlin, (Betreuer: Godbersen) 1984
- [Pa90] Pape, U.:
Qualitätssicherung von CAD-Software.
Berichtskolloquium. 23.4.90, TU Berlin
- [Pf91] Pfafferott, A.:
Softwaresynthese in Verteilten Systemen.
Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen) 1991
- [Ra86] Rastgooy, Trümner, Hänschke:
Eine Entwicklungsumgebung zur Erstellung von CAD-
Anwendungssoftware.
Interner CIS-Bericht 8/86, TU Berlin, Dez. 1986
- [Ra88] Rastgooy, K.; Trümner, H.:
CAD Programm Development Through Construction With Nets.
Interner SFB 203-Bericht, TU Berlin, 1988
- [Ra90] Rastgooy, K.:
Eine Methodenkette zur integrierten Software-Entwicklung - Von der
Problemstellung zum Programmsystem mit Petrinetzen.
Dissertation, TU Berlin, Institut für Angewandte Informatik, 1990
- [Sc80] Schneider, H.-J.; Godbersen, H.P.; Ehmke, I.; Scheschonk, G.:
ABRAHAM/ISAC - The Kernel of an Information System Specification
Method.
Report submitted to the IFIP WG 8.1 meeting, Paris (1980)
- [Sc82] Schneider, H.-J.:
Techniques and Formal Tools for Design, Realization and Evaluation of
Evolutionary Information Systems.

- In: Hawgood (ed.): Proc. IFIP TC-8 Working Conf. on Evolutionary Information Systems, North Holland (1982)
- [Sch83] Schiffner,G.; Scheuermann,P.; Seehusen,S.; Weber,H.:
On a Specification and Performance Evaluation Model for Multicomputer Database Machines.
Proc. 3. Intern. Working Conf. on Database Machines, München, (Sept. 1983)
- [Sch84] Schiffner,G.:
A Specification and Performance Evaluation Modell for Multicomputer Database Machines.
Dissertation, Universität Bremen, 1984
- [Sch86] Schiffner, G.; Godbersen, H.P.:
Function Nets: A comfortable tool for simulating database system architectures.
Simulation, May 1986, pp. 201-210
- [SFB87]
Geometriebezogene Qualitätssicherung in CAD-Softwaresystemen.
in: Spur, G. (Hrsg.): Forschungsbericht 1984-87, Sonderforschungsbereich 203: Rechnergestützte Konstruktionsmodelle im Maschinenwesen., TU Berlin, pp. A5-1 ..97
- [Tr86] Trümner, H.; Rastgooy, K.; Schulze, D.:
Der graphische Netzeditor des FUN- Systems.
TU Berlin, Interner CIS-Bericht 07/86
- [Tr86a] Trümner, H.; Godbersen, H.P.:
Codegenerierung mit Funktionsnetzen.
TU Berlin, Interner CIS-Bericht 06/86
- [Tr88] Trümner, H.:
Funktionsnetze: Ein auf Bausteinen basierender Ansatz zur Software-Konstruktion.
Dissertation, TU Berlin, Institut für Angewandte Informatik, 1988
- [We85] Weber, W.:
Kommunikation in einer Verteilten Software-Entwicklungsumgebung
Kommunikation in Verteilten Systemen, Informatik Fachberichte Nr. ?, 1985
- [Wi86] Winter, D.; Scheschonk,G.; Gier, K.:
ISAC- ein System zur Unterstützung der Systembeschreibung und der Systemanalyse.
in: Handbuch Moderne Datenverarbeitung, Heft 130, 1986, pp. 97-106

[Anfang](#)

7.2 Hintergrundliteratur

- [An91] Adrews, G.R.:
Paradigms for Process Interaction in Distributed Programs.
ACM Computing Surveys, Vol.23, No. 1, March 1991, pp. 49-90
- [ANSA89]
ANSA: An Engineer's Introduction to the Architecture.
TR 03.02. Architecture Projects Management Ltd., Cambridge, UK, 1989
- [Ba87] Bathe, C.-D.; Godbersen, H.P.:
The INCA Workstation.
Proc. ESPRIT Technical Week, North Holland Publ. Co., 1987
- [Bä93] Bässmann, H.; Besslich, Ph.W.:
Bildverarbeitung. Ad Oculos. Springer Verlag. 2. Auflage. 1993
- [Ca90] Carriero, N.; Gelernter, D.:
How to Write Parallel Programms.
The MIT Press, 1990
- [Ch91] Chin, R.S.; Chanson, S.T.:
Distributed Object-Based Programming Systems.
ACM Computing Surveys, Vol.23, No. 1, March 1991, pp. 91-124
- [CMSO92]
Final Report. Esprit II Project 2277. CMSO - CIM for Multi-Supplier
Operations.
CMSO-Consortium, 12.3.92
- [Co94] Coulouris, G.; Dollimore, J.; Kindberg, T.:
Distributed Systems. Concepts and Design.
Addison-Wesley, 2. Edition, 1994
- [De93] Dehnert, E.:
Software-Engineering in Wissenschaft und Wirtschaft: Wie breit ist die
Kluft?
Informatik Spektrum, (1993) 16: 295-299
- [DeJ96] DeJesus, E., Rymer, J.; Salamone, S.; Kador, J.:
The Middleware Riddle. (the Muddle in the Middle. Middle(ware)
Management. The Ultimate Middleware)
in: BYTE April 96.
- [DIN89]
Schnittstellen der rechnerintegrierten Produktion (CIM) - CAD und NC-
Verfahrenskette.
Kommission Computer Integrated Manufacturing (KCIM) (Hrsg.): DIN
Fachbericht 20, Beuth Verlag, 1989
- [DIN90]
Fertigungssteuerung und Auftragsabwicklung..
Kommission Computer Integrated Manufacturing (KCIM) (Hrsg.): DIN

- Fachbericht 21, Beuth Verlag,
 [DoD88]
 Computer-aided Aquisition and Logistic Support (CALs): Program
 Implementation Guide. Department of Defence, Military Handbook #59.
 Washington, DC., 1988
- [ECMA 131]
 Referenced Data Transfer. Standard ECMA-131, July 1988
- [ECMA TR/29]
 OSI Distributed Interactive Processing Environment (DIPE). ECMA
 TR/29, Sept. 1985
- [ECMA TR/31]
 Remote Operations, Concepts, Notation and Connection-Oriented
 Mappings. ECMA TR/31, Dec. 1985
- [ECMA-127]
 RPC. Basic Remote Procedure Call Using OSI Remote Operations.
 Standard ECMA-127, Dec. 1987
- [El82] Ellis, C.A.; Bernal, M.:
 Officetalk-D: An experimental Office Information System. Proc. SIGOA
 Conf., Philadelphia, June 1982
- [EW95]
 Electronics Workbench, Das Elektroniklabor im Computer.
 Demoversion 4.01, Firmenschrift. Interactive Technologies Ltd., Com Pro
 Hard- und Software Vertriebs GmbH., Stuttgart
- [Fa91] Falkenberg, E.D.; van der Pols, R.; von der Weide, Th. P.:
 Understanding Process Structure Diagrams.
 Information Systems, Vol. 16, No.4 pp.417-428, 1991
- [FZK/IAI96]
 Globus III. Abschlußbericht.
 Wiss. Berichte Forschungszentrum Karlsruhe. FZKA-5700. (Vorabversion
 WWW)
- [Go81b] Godbersen, H.P.; Munz, R.; Schneider, H.-J.; Schiele, F.; Steyer, F.:
 Abschlußbericht der Projektes 'Verteiltes Datenbanksystem auf einem
 Kleinrechner-Verbund'.
 BMFT, FKZ 081-5011 A (1981)
- [Go85] Godbersen, H.P.; Malpeli, F.:
 First Results on the Integration of Communication and Application
 Services in the Office.
 Proc.: ESPRIT Technical Week, North Holland Publ. Co. 1985
- [Go86a] Godbersen, H.P.; Köhler, G.; Lewin, Ch.; Seib, J.; Zschoche, G.:
 Bearbeitung, Verwaltung und Versand von Dokumenten.
 Proc. GI-Jahrestagung, Informatik Fachberichte Bd. 126, 1986 p. 118-133
- [Go90a] Godbersen H.; Matthiesen, M.; Schaber, W.:

[Modelling of Interorganisational Operations.](#)

Proc. TELEMATICS '90 Conference, BIBA/Universität Bremen, Bremen
3-5 Dec. 1990, p. 207-221

[Go90b] Godbersen, H.P.; Pavelin, R.:

CMSO Focussing Task Force Report.

ESPRIT II Project 2277: CSMO Report, March 1990

[Goo91] Goose, U.:

Streams - Neue Möglichkeiten der Netzwerkprogrammierung unter UNIX.
Ein Vergleich von Sockets und TLI.

Diplomarbeit, TFH Berlin, (Betreuer: Prof. Godbersen) 1991

[He89] Herrtwich, R.G.; Hommel, G.:

Kooperation und Konkurrenz. Nebenläufige, verteilte und
echtzeitabhängige Programmsysteme.

Studienreihe Informatik, Springer, 1989

[Her89] Hermes, H.:

UN/EDIFACT Weltstandards: Einheitliche Anwendungsschnittstellen für
den elektronischen Datenaustausch.

in: Kühn, P.J. (ed.): Kommunikation in Verteilten Systemen, Begleitband
zum Tutorium anlässlich der GI/TG Fachtagung, Stuttgart, 1989

[Hmä87] Hämmäinen, H.; Eloranto, E.:

Object-Oriented Data Communication for Loosely Coupled Control.

IFIP WG 5.7, Computers in Industry, Vol. 9, No. 4, Dec. 1987 pp. 319-328

[Ho91] Horn, C.; Cahill, V.:

Supporting distributed applications in the Amadeus environment.

computer communications, vol.14 no 6, pp. 358-365, 1991

[HP89]

Task Broker for Networked Environments based on the UNIX Operation
System.

Hewlett Packard, Technical Data, 12/1989

[It88] Itter, F.:

Rechnergestützte Planung kanbangesteuerter Fertigungen - durch
integrierte Systemanalyse und Simulation mit dem PSItool NET.

Interner PSI Bericht (?), Sept. 1988

[Jo96] Joch, A.; Linthicum, D.; Halfhill, T.; Salmone, S.:

Killer Components. (Integration, not Perspiration. Components
Everywhere.)

In: Byte, Jan. 1996

[Ka89] Kramer, J.; Magee, J.; Ng, K.:

Graphical Configuration Programming.

Computer, Oct. 1989, pp. 53-65

[Kre93] Kreifels, T.:

Bürovorgänge: Ein Modell für die Abwicklung kooperativer

- Arbeitsabläufe in einem Bürosystem.
In: Wißkirchen, P. et. al. (eds.): Informationstechnik und Büro, Teubner, Stuttgart, 1983
- [Le89] Leszak, M.; Eggert, H.:
Petri-Netz-Methoden und Werkzeuge. Hilfsmittel zur Entwurfsspezifikation und -validation von Rechensystemen.
Informatik Fachberichte Bd. 197, 1989
- [Li95] Linthicum, D.:
Rapid Application Development (RAD).
in Byte, August 1995
- [Lo90] Lorin, H.:
Application development, software engineering and distributed processing.
computer communications, vol.13, no 1 , pp. 4-15, 1990
- [Lu79] Lundsberg,M.; Goldkuhl,G.; Nilsson,A.:
A Systematic Approach to Information System Development.
Information Systems, Vol.4, p.1-12, 93-118 (1979)
- [Ma91] Matthiesen,M.; Godbersen, H.P.; Schaber, W.:
The CSMO Reference Model for Interorganisational Communications.
Proc. CIM-Europe Conference, Turin, May 29-31, 1991
- [Mo90] Moffett, J.; Sloam, M.; Twidle, K.:
Specifying discretionary access control policy for distributed systems.
computer communications, vol.13, no.9, pp.571-580, 1990
- [Mo91] Moffett, J.; Sloman, M.:
The Representation of Policies as System Objects.
Discussion Paper, Imperial College, London, 16/5/91
- [MS2O94]
Final Report. Esprit III Project 6706. MS²O: Multi-Supplier / Multi-Site Operations.
MS²O Consortium, 10.3.94
- [Mü85] Müller, G.; Proefrock, A.K.:
CLERK - Ein Werkzeugsystem für die Entwicklung objektorientierter Büroanwendungen.
Nixdorf Compuert, Berlin, to appear: GI Nachrichten Interaktive Systeme
- [Nu83] Nutt, G.J.:
An Experimental Distributed Modelling System.
ACM Transactions on Office Information Systems, Vol. 1, No. 2, April 1983, pp. 117-142
- [Ob84] Oberquelle, H.:
Basic Concepts of Object Flow Nets.
Uni Hamburg, IFI-HH-M-122/84
- [Ob85] Oberquelle, H.:
Semi-Formal Graphic Modelling of Dialog Systems.

- Proc. 6. European Workshop on Application and Theory of Petri Nets, Espoo, SF, June 1985
- [Ol83] Olle, T.W.; Sol, H.G.; Tully, C.J. (eds.):
Information Systems Design Methodologies: A Feature Analysis.
Proc. IFIP WG 81 Working Conference, North Holland, 1983
- [Or95] Orfali, R.; Harkey, D.; Edwards, J., Gray, J. Mackenzie, J. DeJesus, E.:
The future of Client/Server Computing. (Intergalactic Client/Server Computing. Scale up with TP Monitors. Document Repositories. Dimensions of Data. Client/Server with Distributed Objects)
in: BYTE April 1995.
- [Po91] Popescu-Zelentin, R.; Tschammer, V.; Tschichholz, M.:
'Y' distributed application platform.
computer communications, vol.14, no 6, pp. 366-374, 1991
- [Pos96] Poswig, J:
Visuelles Programmieren. Hanser Verlag, 1996
- [Re89].Reade, Chris:
Elements of Functional Programming. Addison Wesley Publ. Co. 1989
- [Sc90] Schneider, H.-J.; et.al.:
CIM for Multi-Supplier Operations.
Proc. APMS '90 IFIP TC5/WG5.7 Conf., Espoo, Finland, August 1990
- [Sche89] Scheschonk, G.; Vogt, A.:
Design/OA- Eine Software-Entwicklungsumgebung für visuelle Entwurfssysteme.
Tagungsband 3. Kolloquium "Software-Entwicklungssystem und - Werkzeuge", Esslingen, 1989
- [SG91]
IRIS Explorer: modulatorientiert zur Daten-Visualisierung.
Firmenschrift Silicon Graphics Performer, 6. Ausgabe 91/92
- [Sie96] Siegel, J.:
OMG: Building the Object Technology Infrastructure.
IFIP/IEEE Int. Conf. Distributed Platforms. Tutorial, ICDP 96, Dresden
- [Sie96a] Siegel, J.:
A Tour of CORBA & OMG.
IFIP/IEEE Int. Conf. Distributed Platforms. Tutorial, ICDP 96, Dresden
- [Sl90] Sloman, M.:
Management for Open Distributed Processing.
Second IEEE Computer Society Workshop on Future Trends of Distributed Computer Systems, Cairo, Oct. 1990
- [So88] Sogatz, U.; Hochfeld, H.J.:
Austausch produktdefinierender Daten im Anwendungsgebiet der Karosseriekonstruktion.
Informatik Spektrum, Bd. 8, Heft 6, pp. 305-311, 1988

[SUN]

ToolTalk.

SunSoft ??

[Tuv96] Tuvell, W.:

DCE 1.0 Security Technology -Detailed Architectural Overviews-.

IFIP/IEEE Int. Conf. Distributed Platforms. Tutorial, ICDP 96, Dresden

[Um96] Umar, A.:

Data and Application (Re)Engineering in Client/Server Environments.

IFIP/IEEE Int. Conf. Distributed Platforms. Tutorial, ICDP 96, Dresden

[Va94] Varhol, P.:

Visual Programming's Many Faces.

In: Byte, July 1994

[Wa96] Warnke, K.:

Verteilte Systeme mit ToolTalk.

Diplomarbeit TFH Berlin, FB Informatik (Betreuer: Prof. Godbersen),

1996

[Wel95] Wells, L.:

LabVIEW Student Edition User's Guide.

Prentice Hall, 1995

Copyright © Godbersen, 1996 [Anfang](#) 29.09.96 7.htm