KfK 2507
Januar 1978

Prüfmittel für die rechnergestützte Software-Entwicklung

J. Ludewig, W. Streng
Institut für Datenverarbeitung in der Technik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Datenverarbeitung in der Technik

KfK 2507

PRÜFMITTEL FÜR DIE RECHNERGESTÜTZTE SOFTWARE-ENTWICKLUNG

J. Ludewig
W. Streng

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Zusammenfassung:

Ausgehend von einer allgemeinen Charakterisierung einer bestimmten Klasse von Entwicklungssystemen wird ein Ansatz für ein einfaches Beschreibungsmodell für Systeme angegeben. Ein solches Modell ist die Voraussetzung, um die Beschreibung der Anforderungen und des Entwurfs eines Systems zu formalisieren.

Dieses Modell bildet die Grundlage für den dann folgenden Vergleich der Beschreibungsmodelle und Prüfmittel dreier bekannter Entwicklungssysteme: SSES, PSL/PSA und SREP.

Abschließend wird die Schnittstelle derartiger Systeme zu einem automatischen Testsystem diskutiert.

Abstract: Validation Tools for Computer-Aided Software Development

Starting with a general characterization of a particular class of software development systems, an approach for a single system description model is presented. Such a model will serve as the basis for formalizing the description of the requirements and the design of a system.

In this paper the model is used for comparing the description models and checking facilities of three existing development systems: SSES, PSL/PSA and SREP.

Finally the interface between those systems and an automatic testing system is discussed.

I n h a l t :

1. Einleitung

Aufgrund von Fehleranalysen und Erfahrungen bei der Entwicklung großer Software-Systeme hat sich inzwischen die Einsicht verbreitet, daß eine Erhöhung der Zuverlässigkeit vor allem durch eine Formalisierung der Spezifikation und durch eine Methodisierung des Entwurfs zu erreichen ist. Erst durch formalisierte Beschreibungshilfsmittel wird eine rechnergestützte Software-Entwicklung und -Prüfung ermöglicht. Bisher ist der Anwender bei der Formulierung seiner Anforderungen auf natürliche Sprachen angewiesen. Dies führt häufig zu unvollständigen, mehrdeutigen und nicht selten widerspruchsvollen Spezifikationen, die dann die Basis für die Software-Entwicklung bilden. Enthält ein System systematische Fehler infolge einer falsch erkannten oder formulierten Aufgabenstellung, so nützen auch ausführliche Korrektheitsprüfungen nichts.

Um zu prüfbaren Spezifikations- und Entwurfsbeschreibungen zu gelangen, wurden rechnergestützte Software-Entwicklungssysteme entworfen, die auf einem formalen Systembeschreibungsmodell beruhen.

Angeregt durch eine Diskussion über die Weiterentwicklung des Testsystems SADAT*) sollen Prüfmittel für Spezifikation und Entwurf und deren Schnittstellen zu einem derartigen Testsystem diskutiert werden. Um möglichst konkret zu sein, soll dies auf der Basis existierender rechnergestützter Entwicklungssysteme geschehen.

Zu diesem Zweck werden zu Beginn die Grundkonzepte**) der drei Entwicklungssysteme SSES, PSL/PSA, SREP aufgezeigt und ihre Prüfmittel abgeleitet. Davon ausgehend werden mögliche Erweiterungen des Test-

---

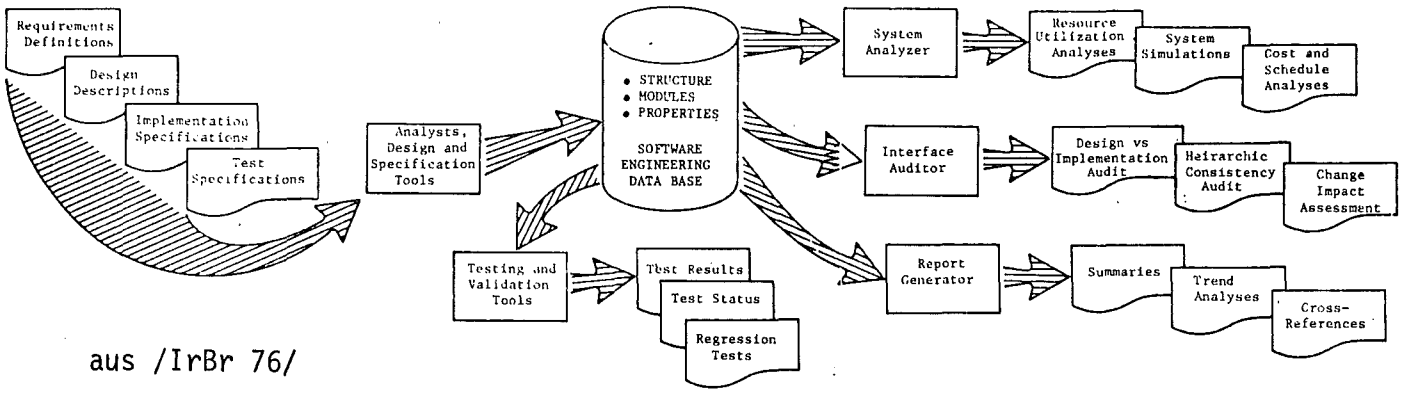
*) Vgl. /Sei 76/.

**) Eine ausführlichere Beschreibung und ein Vergleich mit anderen Systemen ist im Bericht KFK 2506 enthalten.

systems vorgeschlagen. Bei einem derartigen Vorgehen wird immer wieder die Forderung nach der Berücksichtigung von Anwenderwünschen gestellt. Es hat sich jedoch gezeigt, daß die Anwender nicht in der Lage sind, ihre Anforderungen zu spezifizieren. Um sich dennoch nicht zu weit von der Praxis zu entfernen, bleibt nur die Möglichkeit, die Anwender zur Benutzung und Erprobung bereits fertiggestellter Produkte zu motivieren (mit dem Essen kommt der Appetit) und sie dann, ausgerüstet mit diesen Erfahrungen, an der Weiterentwicklung zu beteiligen.

2. Rechnergestützte Software-Entwicklungssysteme

Wir beschränken uns hier auf Systeme, die man den sogenannten Datenbank-orientierten Systemen zurechnen kann. Angestrebt wird dabei jeweils ein integriertes Werkzeug von aufeinander abgestimmten Hilfsmitteln zur Überdeckung des sogenannten Entwicklungszyklus. Das Ziel ist, alle in den einzelnen Entwicklungsschritten gewonnenen Informationen in maschinenverarbeitbarer Form darzustellen und in einer Datenbank zur weiteren Bearbeitung (Prüfung) abzuspeichern. Ein solches System könnte die folgenden Komponenten enthalten:



aus /IrBr 76/

Beschreibungsteil Datenbank Analyseteil

Dieses Konzept hat drei wesentliche Eigenschaften:

- Unterstützung der entwicklungsbegleitenden Dokumentation. Durch ein geeignetes Datenbankmanagementssystem ist die Dokumentation leicht anpaßbar, änderbar, erweiterbar und stets auf dem aktuellen Stand. Es gibt nur eine Dokumentation, auf die sich alle Entwickler beziehen und über die sie kommunizieren.

- Trennung zwischen Beschreibungsteil (Bereitstellung von Informationen) und Analyseteil (Prüfung der Informationen), wobei die DB die Schnittstelle bildet. Diese Entkopplung erleichtert Änderungen der beiden Teile.

- Durch die maschinenverarbeitbare Darstellung der Ergebnisse der einzelnen Entwicklungsschritte können Prüfungen auf Vollständigkeit, Eindeutigkeit und Widerspruchsfreiheit rechnergestützt durchgeführt werden. Fehler der Aufgabenstellung und des Entwurfs können frühzeitig (d.h. kurz nach dem Zeitpunkt, zu dem sie entstehen) erkannt und beseitigt werden. Fehleranalysen haben gezeigt, daß gerade Fehler, die durch eine falsche Interpretation der Problemanalyse oder aber auf eine falsche oder unvollständige Problemanalyse zurückgehen, besonders häufig und schwierig zu entdecken sind (nach der Implementierung).

3. Ein einfaches Systembeschreibungsmodell (SBM)

3.1 Definition eines SBM (informell)

Voraussetzung für die Entwicklung einer formalen Sprache zur Beschreibung der Anforderungen an ein System ist ein Modell für das zu beschreibende System. Geht man von einem ganz allgemeinen Systembegriff aus, so erhält man ein ganz einfaches Modell.

Das Modell sagt aus, daß ein System aus handhabbaren Dingen besteht, die Objekte genannt werden. Einem Objekt können ein Name und ein Typ zugeordnet werden. Zur näheren Darstellung der Objekte muß das Modell Möglichkeiten enthalten, die jeweiligen spezifischen Eigenschaften zu beschreiben. Objekte können folglich Attribute besitzen; jedes dieser Attribute hat einen speziellen Attributwert aus einem definierten Wertebereich. Objekte können auf vielfältige Weise miteinander in Beziehungen stehen; diese werden durch die Angabe von Relationen beschrieben (die Relationen sind nicht auf binäre Relationen beschränkt).

Eine bestimmte Klasse von Systemen kann durch Spezialisierung dieses allgemeinen relationellen Modells charakterisiert werden, indem man die verfügbaren Objekte, Attribute und Relationen festlegt. Da noch keine allgemeine Theorie hierfür existiert, wird man sich dabei einerseits auf Erfahrungen stützen, die man bei der informellen Systemdefinition gemacht hat, andererseits auf Abstraktionen, die aus der Implementierung von Systemen ableitbar sind. Wir wollen den Informationsraum, der von einem Beschreibungsmodell aufgespannt wird, in Klassen einteilen und diese Klassen Systemaspekte nennen. Jedem Systemaspekt wird eine Menge von Objekten und Relationen in dem Modell zugeordnet. Sind die Objekte z.B. Komponenten (Moduln) eines Soft-

ware-Systems, so lassen sich ganz grob die beiden Systemaspekte statisch-funktional und dynamisch-leistungsorientiert (als Beziehungen zwischen diesen Komponenten) unterscheiden.

Eine weitere Verfeinerung des Modells erreicht man, indem man Beschreibungsebenen einführt, d.h. verschiedene Abstraktionsebenen der Beschreibung (Strukturen).

Die Mächtigkeit eines Beschreibungsmodells ergibt sich aus der Anzahl der Systemaspekte, die beschrieben werden können, und aus ihrer Adäquatheit für eine bestimmte Klasse von Systemen (Anwendungsgebiet).

3.2 Syntaktische Repräsentation eines SBM

Nachdem man sich Klarheit verschafft hat über die zu beschreibenden Informationen, muß man eine formalisierte Sprache entwickeln, in der die in dem Beschreibungsmodell enthaltenen Informationen ausgedrückt werden können. Wir wollen eine solche Sprache Spezifikationssprache (SPESP) nennen. Aus dem Beschreibungsmodell läßt sich ableiten, daß eine SPESP vier primitive Sprachelemente enthalten kann: Objekte, Relationen, Attribute und Strukturen. Diesen wird eine definierte Syntax zugeordnet. Damit kann der Anwender die SPESP benutzen, um seine Problembeschreibung präzise zu formulieren und in einer DB abzuspeichern. Letzteres wird von einem Sprachprozessor bewerkstelligt, der die Sprachkonstruktionen in SPESP in geeignete Konstruktionen der DB abbildet. Dabei führt er eine Syntaxanalyse durch und überprüft die Eingabe mit den schon in der DB enthaltenen Konstruktionen auf Widerspruchsfreiheit. Der Inhalt der DB kann als abstrak-

tes Modell der Problembeschreibung aufgefaßt werden: In diesem Modell werden Objekte durch Knoten (records) in der DB dargestellt, während Zeiger die Relationen zwischen ihnen repräsentieren. Attribute und ihre Werte bestehen aus einem Knoten für den Wert und einem Zeiger zum Objektknoten, dem dieses Attribut zugeordnet ist. Strukturen werden in den Graphen expandiert, den sie darstellen.

Auf der Grundlage einer so definierten Spezifikationssprache lassen sich einige prüfbare Eigenschaften präziser formulieren:

- Widerspruchsfreiheit (Konsistenz): Keine Aussagen in der SPESP-Beschreibung widersprechen sich; auf ein bestimmtes Objekt wird in der gesamten Beschreibung unter dem gleichen Namen Bezug genommen. Der erzeugte Graph ist zusammenhängend.

- Vollständigkeit : Alle notwendigen Relationen sind definiert und keine Objekte fehlen.

- Eindeutigkeit : Gemäß der Semantik von SPESP ist die Interpretation der Beschreibung für alle Leser und zu allen Zeitpunkten gleich.

4. Beschreibungsmodelle und Prüfmittel der drei betrachteten Entwicklungssysteme

Wie schon in der Einleitung erwähnt, sollen hier die Grundkonzepte der Entwicklungssysteme SSES, PSL/PSA und SREP und deren Prüfmittel vorgestellt werden. Diese drei Systeme basieren auf der im vorigen Kapitel angegebenen Philosophie eines relationellen Beschreibungsmodells. Gemeinsam haben sie eine formale Beschreibungssprache und eine Datenbank. Die Mächtigkeit ihrer Beschreibungsmodelle ist jedoch unterschiedlich (SSES $\subset$ PSL/PSA $\subset$ SREP).

4.1 SSES - Software Specification and Evaluation System

/Au 76/, /HoRy 76/

4.1.1 Ziele des SSES

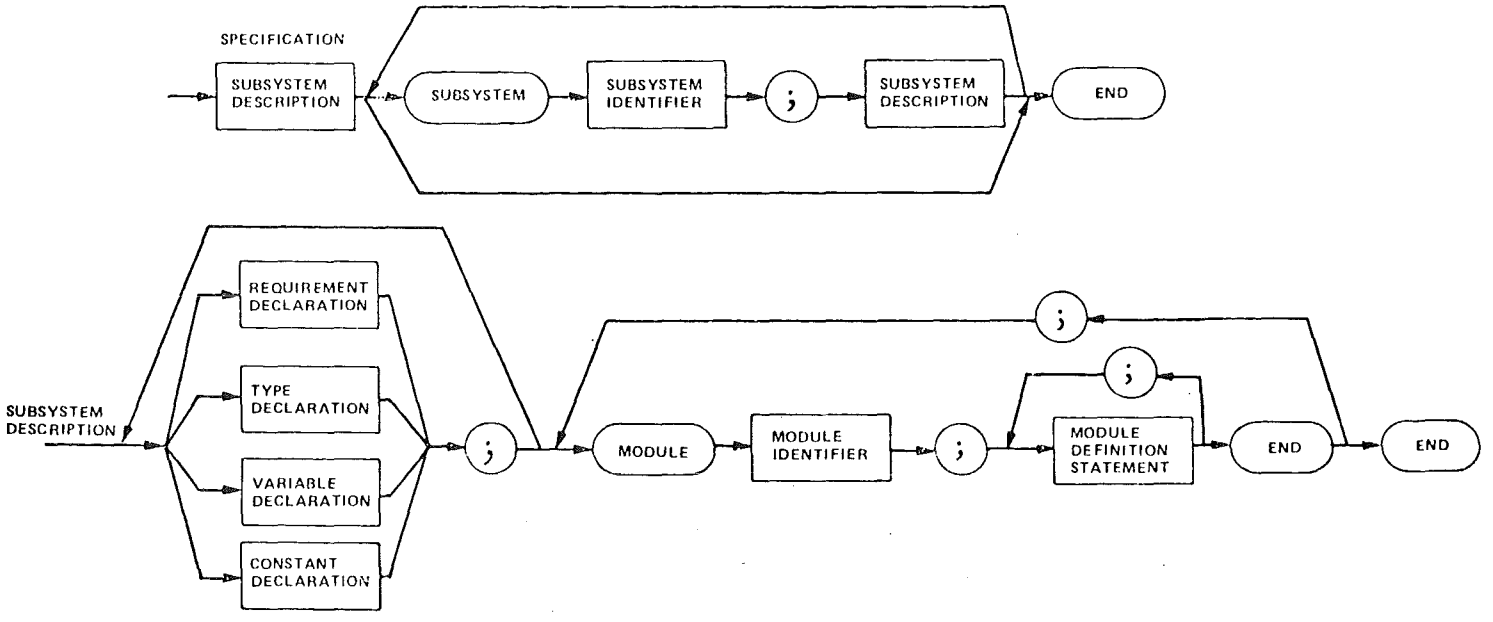
Eine Software-Spezifikation soll:

- formal sein
- die Zuverlässigkeit verstärken
- vollständig, eindeutig, widerspruchsfrei und testbar sein
- eine gleichmäßige Abstraktionshöhe haben
- maschinenunabhängig sein
- Schnittstellen einfach und klar beschreiben
- leicht änderbar sein
- leicht verwendbar sein
- Anforderungen an Fehlerbehandlung und -toleranz darstellen
- die ursprünglichen Anforderungen erkennen lassen, rückverfolgbar sein

4.1.2 Beschreibungsmodell des SSES

Das Beschreibungsmodell von SSES kann man als Entwurfsmodell charakterisieren. Die beschreibbaren Systemaspekte beschränken sich auf Informationen, wie sie in einem funktionellen Blockdiagramm eines Software-Systems enthalten sind (statisch-funktionelle Struktur). Darüberhinaus können Datenstrukturen, Modulschnittstellen und Ein-/Ausgabe-Variablen eines Moduls beschrieben werden. Wesentliches Objekt der Beschreibung ist der Modul. Als Struktur (Abstraktionsebene) gibt es noch Subsysteme, bestehend aus einem oder mehreren Moduln.

Nachfolgend ist die Grammatik für Subsysteme dargestellt:



Hervorzuheben sind noch zwei weitere Systemaspekte:

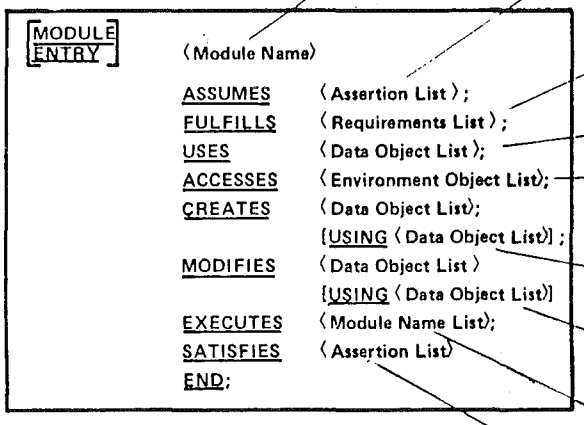
- Rückverfolgbarkeit der Anforderungen:

  Anforderungen können bestimmten Objekten zugeordnet werden, und während des Zerlegungsprozesses wird über sie Buch geführt

- Dynamische Eigenschaften eines Objektes:

  Durch die Definition von Zusicherungen (Assertions) kann das dynamische Verhalten beschrieben werden
  (z.B. a[i]=0 FOR ALL i=[1...n], code=1 IMPLIES (eof EQU TRUE))

Die syntaktische Repräsentation des Beschreibungsmodells wird durch die nicht prozedurale Spezifikationssprache SSL definiert. Als Beispiel sei die syntaktische Repräsentation der Relationen zur Beschreibung eines Moduls in SSL angegeben:



Identifikation eines neuen Moduls durch seinen Namen, es wird zwischen zwei Modultypen unterschieden.

Angabe von Zusicherungen (assertions), die vor Modulausführung gültig sein müssen.

Aufzählung der Anforderungen, die durch diesen Modul realisiert werden.

Auflisten der Eingabedaten

Auflisten von Elementen der Umgebung, die benutzt werden.

Ausgabedaten, die initialisiert werden.

Transiente Daten, die verändert werden.

Angabe der aufrufbaren Moduln.

Angabe von Zusicherungen (assertions), die nach Modulausführung gültig sein müssen.

Beispiel einer Modulbeschreibung in SSL:

```
MODULE SORT  (N:INTEGER) ;
   /*  MODULE TO SORT ARRAY */
   /*  ARRAY IS INITIALIZED FROM CARD READER  */

   ASSUMES          N  >  0 ;
   FULFILLS         ORDERED _ VALUE;
   ACCESSES         CARD_READER;
   MODIFIES         SARRAY USING N;
   SATISFIES        FORALL  (I:INTEGER)
                    I  >  0  AND  I  ≤  N-1
                    AND
                    SARRAY [I]  ≥ SARRAY  [I+1]

END;
```

4.1.3 Prüfmittel von SSES

- Automatisches Testsystem
  Die Eingabe für das Testsystem besteht aus Fortran-Quellcode, der durch einen im System enthaltenen Preprozessor für strukturiertes FORTRAN erzeugt wurde. Um aus der Entwurfsbeschreibung in SSL gut strukturierte Programme zu entwickeln, wurde eine geeignete Fortran-Erweiterung definiert. Ein wesentliches Kriterium für die Erweiterung war die leichtere Analysierbarkeit des erzeugten Codes.

  - Statische Analyse
    Analyse des Quellcodes auf Element-, Ausdruck-, Statement-, Modul- und Programmebene.

  - Dynamische Analyse
    - Testüberdeckung für einen Testlauf
    - Teststatistiken
    - Testgeschichte

  - Testdatenerzeugung
    - Schätzung der Gesamtzahl von ausführbaren Pfaden
    - Minimale Überdeckung
    - Schätzung aller "wichtigen" Pfade

    Obwohl das Problem der Erzeugung von Eingabedaten für den Test eines vorgegebenen Programmstücks allgemein nicht entscheidbar ist, sollen heuristische Verfahren hierfür in SSES enthalten sein. Unterstützung ergibt sich auch durch symbolische Testverfahren.

- Unterstützung der Testdatenerzeugung durch SSL
  Eine wesentliche Eigenschaft von SSL ist die Zuordnung der Anforderungen zu den Moduln, die diese Anforderung realisieren (Rückverfolgbarkeit). Wählt man eine Anforderung aus, so ist man durch SSL in der Lage, alle Moduln, die diese Anforderung erfüllen, zu identifizieren. Durch diese problembezogene Klassi-

fizierung von Programmteilen, u.U. mit Zuhilfenahme der in ihnen enthaltenen Zusicherungen, läßt sich die Testdatenerzeugung vereinfachen.

- SSL-Vergleichsdaten für den Test

  - Die Ergebnisse der statischen Code-Analyse werden mit der funktionellen Spezifikation in SSL verglichen (in DB):

    Modulschnittstellen,
    Benutzung von Variablen (Typ, Wertebereiche, cross-reference-listing), Initialisierungen von Variablen oder Feldern, Ein-/Ausgabe Beziehungen.

4.1.4 Beispiel: Telegrammverarbeitung /Au 76/

The example of this section was selected to demonstrate both the descriptive level of SSL and as many language elements as possible. The requirement of the problem may be stated as follows :

> "A program is required to process a stream of telegrams. This stream is available as a sequence of letters, digits and blanks on some device and can be transferred in sections of predetermined size into a buffer where it is to be processed. The words in the telegram are separated by sequences of blanks and each telegram is delimited by the word 'ZZZZ'. The stream is terminated by the occurrence of the empty telegram, that is a telegram with no words. Each telegram is to be processed to determine the number of chargeable words and to check for occurrences of overlength words. The words 'ZZZZ' and 'STOP' are not chargeable and words of more than twelve letters are considered overlength. The result of the processing is to be a neat listing of the telegrams, each accompanied by the word count and a message indicating the occurrence of an overlength word."

To complete the problem statement, several assumptions are necessary. The following alternatives were selected for the purpose of this exposition:

- The character stream from which the telegrams are constructed resides on a drum having fixed length records; the record length itself is left as an implementation option.
- The chargeable word count is the value to be printed and overlength words count as one word.
- If a physical end of file is encountered before the logical end of the data stream, an error message and the partial telegram is printed.

The software is organized into four modules as indicated by Figure 4.1.4a. The purpose of each module is given in Table a.

Figure 4.1.4b contains the SSL description of the telegram processor. The right margin of the statement listing contains reference notes to subsections containing detailed descriptions of the language elements used.

A careful examination of Figure 4.1.4b will indicate an interesting application of the subsystem capability. The subroutines GET_CHAR and FILL_BUFFER occupy a separate subsystem with the sole purpose of handling file I/O. The characteristics of the device on which the telegrams are stored are encapsulated within these two modules.

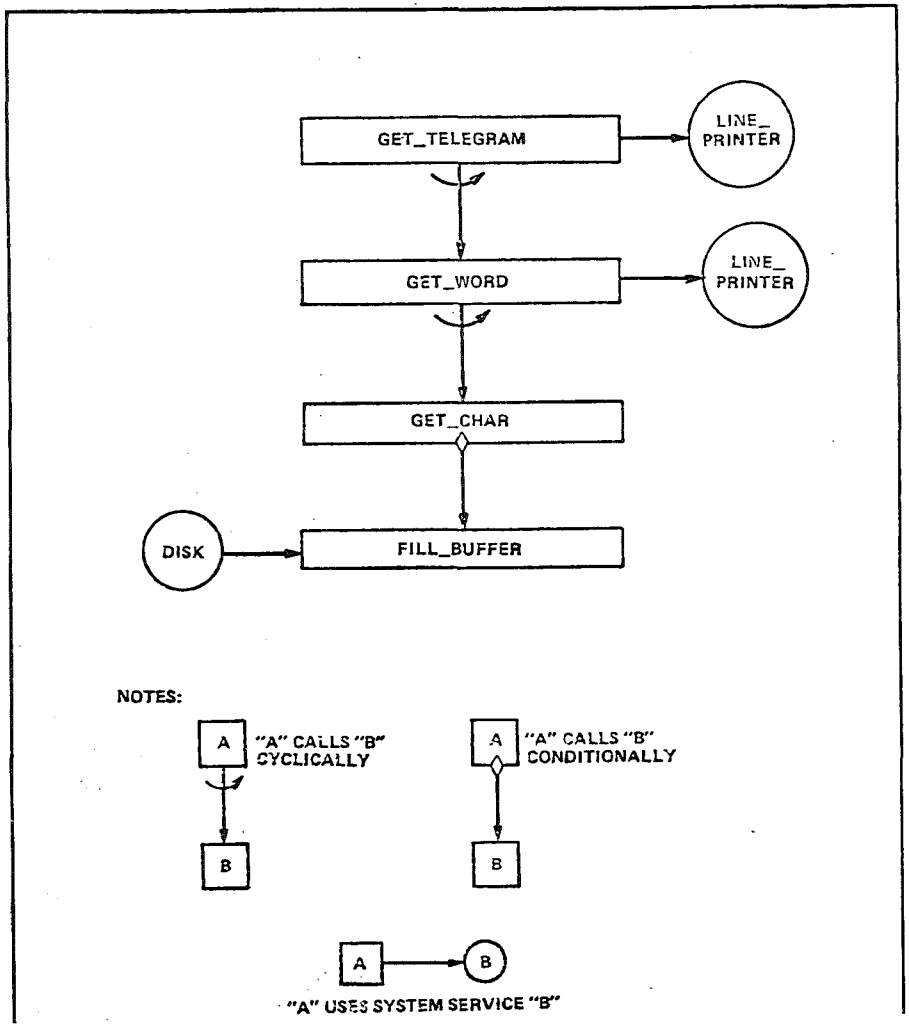


Figure 4.1.4 a: Module Structure Chart for Example

TABLE MODULE DESCRIPTIONS FOR EXAMPLE

MODULE	PURPOSE
GET_TELEGRAM	Collects words belonging to each telegram and prints them in a neat manner along with the chargeable word count.
GET_WORD	Collects characters into words and prints error messages denoting over-length word or physical record end of file.
GET_CHAR	Returns the next character in the telegram file.
FILL_BUFFER	Enters the next physical record from the drum into the character buffer.

```
/* beginning of main subsystem preamble */

requirement
        transductions
        collect in print;
        output
        telegram, charge_count
        end;
variable telegram:text;
        charge_count:integer:
             for print;
             subjectto charge_count ≥ 0
        word_count:integer;
             for print;
             subjectto word_count ≥ charge_count;
        word:array [1..12] of char;
             for print;
        eof_flag:boolean;
             for print
end; /* end of main subsystem preamble */

/* main routine to collect words and */
/* print telegram with chargeable word count */
module get_telegram;
        fulfills print;
        creates telegram, charge_count using word;
        creates word_count;
        modifies word_count;
        uses eof_flag;
        accesses line_printer;
        executes cyclically get_word;
        satisfies
             eof_flag or word_count = 0
        end;




/* subroutine to collect characters into */
/* words */
module get_word;
        fulfills collect;
        executes cyclically i_o.get_char(a_char:char:eof_flag);
        creates word, eof_flag;
        accesses line_printer /*prints error messages */
        end
end; /* end of main subsystem */
```

Figure 4.1.4 b: SSL Description for Example

```
/* beginning of i_o subsystem preamble */

subsystem i_o;

requirement
        input character_file;
        transductions
        read in separate;
        output a_char, eof_flag
        end;

/*  parameterize record length */

constant record_length = integer;

type character_record = array [1..record_length] of char;

variable character_file:sequence of character_record;
                for read;
        buffer:character_record;
                for separate;
        a_char:char;
                for separate;
        char_index:1..record-length;
                for separate;
        eof_flag:boolean;
                for separate

end: /* end of subsystem preamble */



/* subroutine to fetch next */

/* character from file */

entry get_char (a_char; eof_flag);
        fulfills separate;
        executes conditionally fill_buffer;
        modifies char_index;
        creates a_char using buffer [char_index], eof_flag;
        creates character_file, char_index;
        satisfies eof_flag implies a_char = buffer [char_index],
        end;

/* subroutine to fetch next physical */

/* record from character file */

module fill_buffer;
        fulfills read;
        assumes char_index = record_length;
        accesses disk;
        creates buffer, eof_flag using character file ;
        satisfies
            eof_flag implies buffer = character_file
        end

end /* end of subsystem */
end; /* end of specification */
```

Figure 4.1.4 b: SSL Description for Example (continued)

4.2 PSL/PSA - Problem Statement Language/Problem Statement Analyzer

/TeHe 76/

4.2.1 Ziele von PSL/PSA

Entwicklung von Methoden zur Automatisierung (so weit wie möglich) der Erstellung von Informationssystemen. Dabei sollte in folgenden Entwicklungsschritten vorgegangen werden:

- Klärung, welche Informationen bereitgestellt werden müssen
- Entwicklung einer Sprache, in der man diese Informationen ausdrücken kann
- Entwicklung eines Systems zur Speicherung dieser Informationen in einer DB
- Bereitstellung von Prüfmitteln (Widerspruchsfrei, vollständig, eindeutig) und
- Hilfsmitteln zur Entscheidungsfindung

Durch rechnergestützte Hilfsmittel sollen nicht nur Analysen ermöglicht und Änderungen erleichtert, sondern auch die Produktivität der Systemanalytiker, Entwerfer und Programmierer erhöht werden.

4.2.2 Beschreibungsmodell von PSL/PSA

Im Vergleich zu SSES bringt PSL/PSA eine Erweiterung der Objekttypen in Richtung auf die Problemstellung hin wie z.B. PROCESS, INTERFACE, EVENT etc. und vor allem eine Ausweitung der Systemaspekte in funktioneller Hinsicht, in Richtung Datenfluß-Beschreibung und - in geringem Maße - Performance-Beschreibung.

Die Relationen lassen sich durch folgende Systemaspekte klassifizieren:

Systemaspekt		Beschreibung der
• System Flow	:	Eingabe/Ausgabe-Schnittstelle des Systems zur Umgebung oder anderen Teilsystemen
• System Structure	:	Statische Zerlegungsstruktur von Objekten, z.B. Modulhierarchie
• Data Structure	:	Beziehungen zwischen Daten aus Anwendersicht
• Data Derivation	:	Datentransformationen durch Prozesse auf verschiedenen Abstraktionsebenen
• System Size/Volume	:	Parameter, die die Größe des Systems beschreiben und den Aufwand an Berechnungen beeinflussen
• System Dynamics	:	Zeitverhalten des Systems; unter welchen Bedingungen treten Ereignisse ein und was geschieht, wenn sie eintreten
• System Properties	:	Charakteristische Eigenschaften eines Objektes, die es von einem Objekt des gleichen Typs unterscheiden, z.B. Synonyme, informelle Textbeschreibung eines Objektes, Kommentare
• Projekt Management	:	Informationen, die für den Projektverlauf von Bedeutung sind, z.B. Zeitpläne, Verantwortlichkeit von Personen für bestimmte Teilaufgaben

Die syntaktische Repräsentation des Beschreibungsmodells wird durch die nicht prozedurale Spezifikationssprache PSL definiert. Als Beispiel wird ein Ausschnitt aus der Beschreibung eines Lohnabrechnungssystems angegeben:

```
PROCESS                                          HOURLY-EMPLOYEE-PROCESSING;
    DESCRIPTION;
         THIS PROCESS PERFORMS THOSE ACTIONS NEEDED TO INTERPRET
         TIME CARDS TO PRODUCE A PAY STATEMENT FOR EACH HOURLY
         EMPLOYEE.;
    KEYWORDS:       INDEPENDENT;
    ATTRIBUTES ARE:
         COMPLEXITY-LEVEL
                    HIGH;
PART OF:         PAYROLL-PROCESSING;
SUBPARTS ARE:    HOURLY-PAYCHECK-VALIDATION,
                 HOURLY-EMP-UPDATE,
                 H-REPORT-ENTRY-GENERATION,
                 HOURLY-PAYCHECK-PRODUCTION;
RECEIVES:        TIME-CARD;
GENERATES:       PAY-STATEMENT,
                 ERROR-LISTING,
                 HOURLY-EMPLOYEE-REPORT;
DERIVES:         PAY-STATEMENT;
DERIVES:         ERROR-LISTING;
DERIVES:         HOURLY-EMPLOYEE-REPORT;
USES:            TIME-CARD,
                 HOURLY-EMPLOYEE-RECORD;
PROCEDURE;
     1. COMPUTE GROSS PAY FROM TIME CARD .
     2. COMPUTE TAX FROM GROSS PAY.
     3. SUBTRACT TAX FROM GROSS PAY TO OBTAIN NET PAY.
     4. UPDATE HOURLY EMPLOYEE RECORD ACCORDINGLY.
     5. UPDATE DEPARTMENT RECORD ACCORDINGLY.
     6. GENERATE PAYCHECK.
    NOTE: IF STATUS CODE SPECIFIES THAT THE EMPLOYEE DID NOT WORK
       THIS WEEK, NO PROCESSING WILL BE DONE FOR THIS EMPLOYEE
       RECORD.;
TERMINATION-CAUSES:
                 NEW-EMPLOYEE-PROCESSING-INIT;
TRIGGERED BY:    HOURLY-EMP-PROCESSING-INIT;
HAPPENS:
     ONE TIMES-PER  WEEK;
SECURITY IS:     COMPANY-ONLY;
```

Auf zwei Eigenschaften von PSL sei in unserem Zusammenhang hingewiesen:

- So, wie in einer höheren Programmiersprache bestimmte Operationen nur zwischen bestimmten Datentypen definiert sind, hängen in PSL die erlaubten Relationen zwischen Objekten vom jeweiligen Typ dieser Objekte ab. Unverträglichkeiten können automatisch entdeckt werden.

- Die meisten binären Relationen können aus beiden Blickrichtungen beschrieben werden, z.B. "A generates B" oder "B is generated by A". Für den erzeugten Inhalt der Datenbank gibt es dazwischen keinen Unterschied. In den Reports werden beide Aspekte ausgegeben.

Durch die Beschreibung des Objektes data-acquisition

PROCESS data-acquisition;

RECEIVES checkpoint-array;
TRIGGERED BY sensor-stimuli;
UPDATES protocol;

werden bei der Ausgabe der folgenden Objekte die angegebenen komplementären Relationen aufgeführt:

INPUT checkpoint-array;

RECEIVED BY data acquisition;
EVENT sensor-stimuli;
TRIGGERS data acquisition;
SET protocol;
UPDATED BY data acquisition.

4.2.3 Prüfmittel von PSL/PSA

Die Prüfungen einer in PSL formulierten Systembeschreibung führt der "Problem Statement Analyzer" (PSA) durch. Er enthält auch den PSL-Sprachprozessor, der den Inhalt der Datenbank aufbaut. Neben der syntaktischen Prüfung von PSL-Statements führt er dabei auch Korrektheits- und Konsistenzprüfungen der neu einzutragenden Daten mit den bereits in der DB befindlichen Informationen durch.

Mit Hilfe eines Dialogsystems lassen sich verschiedene Änderungen und Analysen durchführen, die wie folgt zu klassifizieren sind:

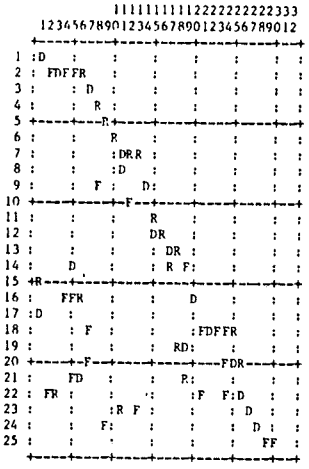
- Datenbankmodifikationen : Protokolle über Änderungen der DB.
- Datenbankinhalte : Ausgabe von DB-Informationen in verschiedenen Formaten;
  z.B. Namenslisten, die alle Objekte in der DB zusammen mit ihrem Typ und dem Datum der letzten Änderung enthalten, Ausgabe aller Eigenschaften und Relationen eines bestimmten Objektes etc. (Reports die verschiedene Systemaspekte beschreiben).

- Statistiken : Ausgabe von Häufigkeiten der Objekte von einem bestimmten Typ; Strukturinformationen die Aussagen über hierarchische Beziehungen zwischen Objekten enthalten; Ausgabe des Datenflußes in graphischer Form.

- Prüfungen : Namenskonflikte können entdeckt werden; Überdeckungen von Ein- und Ausgaben werden untersucht; "Lücken" im Informationsfluß (Daten, die nicht erzeugt werden, Prozesse die keine Daten erzeugen, Daten, die nicht benutzt werden) werden aufgedeckt; das dynamische Verhalten des Systems kann durch Prozeßverfolgungen graphisch sichtbar gemacht werden (Bild 4.2.3 a, b).

Bild 4.2.3 a

DATA PROCESS INTERACTION MATRIX

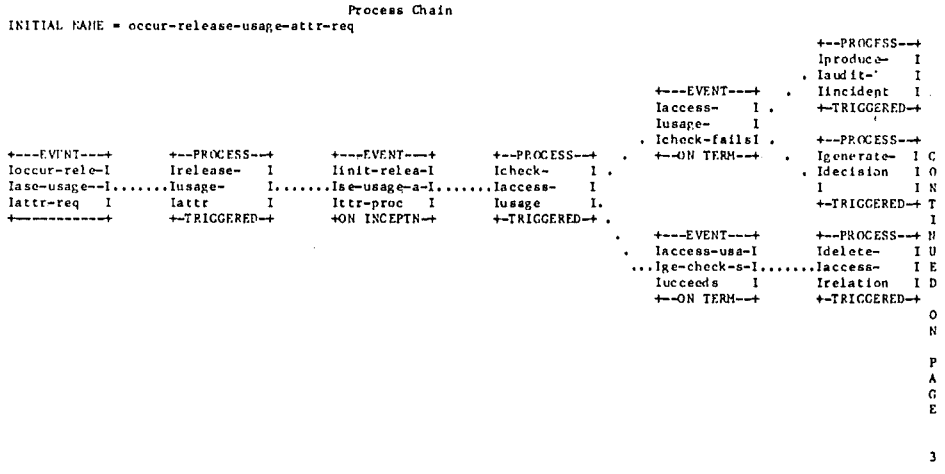


Legende: Die Zeilennummern bezeichnen Datennamen,
die Spaltennummern bezeichnen Prozessnamen.

Die (i, j)-Werte der Matrix haben folgende Bedeutung:

- R - Row i is received or used by column j (input)
- U - Row i is updated by column j
- D - Row i is derived or generated by column j (output)
- A - Row i is input to, updated by, and output of column j (all)
- F - Row i is input to and output of column j (flow)

Bild 4.2.3 b

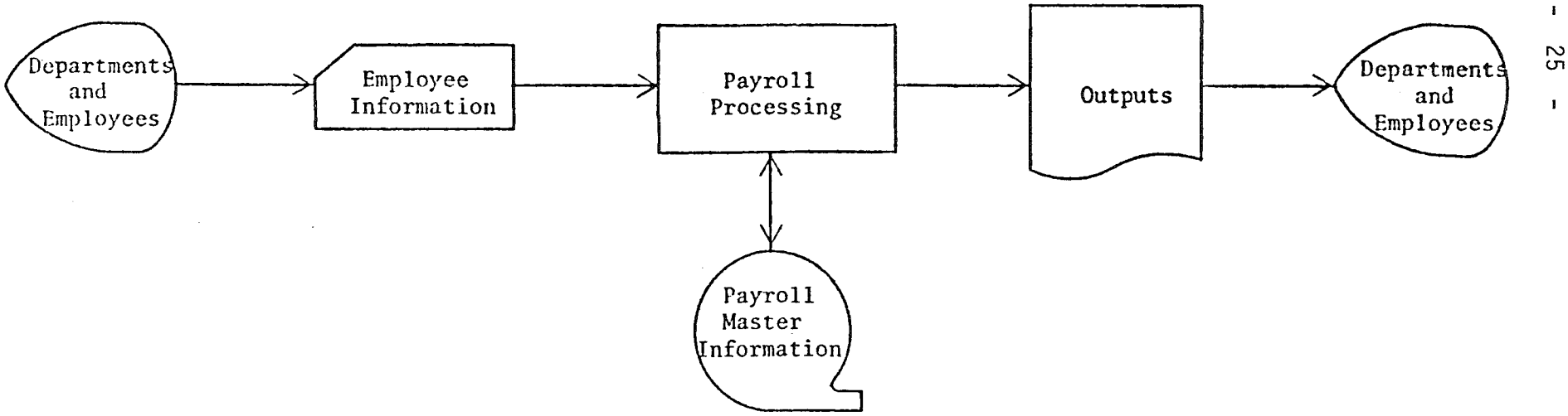


Dieser Report beginnt mit dem gewünschten Objekt und verfolgt alle definierten Prozeß-Objekt-Prozeß Beziehungen (dynamische Relationen). Der EVENT 'occur-release-usage-attr-reg' TRIGGERS den PROCESS 'release-usage-attr', der, wenn er aktiviert wird, den EVENT 'init-release-usage-attr-proc' erzeugt, etc.

4.2.4 Beispiel

Das Beispiel enthält die Eingabebeschreibung in PSL für einen Teil eines Lohnabrechnungssystems mit den folgenden allgemeinen Anforderungen:

"A system called payroll processing takes employee information which comes from departments and employees and produces outputs which go to the departments and employees. The system also maintains payroll master information."



Die Entwurfsbeschreibung enthält nur die drei Systemaspekte System Flow, Communication Aids und System Structure.

```
/* SYSTEM FLOW   */

INPUT: EMPLOYEE-INFORMATION;

OUTPUT: PAYSYSTEM-OUTPUTS;

SET: PAYROLL-MASTER-INFORMATION;

REAL-WORLD-ENTITY: DEPARTMENTS-AND-EMPLOYEES;
     GENERATES: EMPLOYEE-INFORMATION;
     RECEIVES: PAYSYSTEM-OUTPUTS;

PROCESS: PAYROLL-PROCESSING;
     UPDATES PAYROLL-MASTER-INFORMATION;
     RECEIVES: EMPLOYEE-INFORMATION;
     GENERATES: PAYSYSTEM-OUTPUTS;


/* COMMUNICATION AIDS   */

INPUT: EMPLOYEE-INFORMATION;
     SYNONYM: EMP-INFO,I1;
     DESCRIPTION;
          THIS INPUT REPRESENTS ALL THE NECESSARY INFORMATION TO
          PRODUCE THE OUTPUTS FROM THE PAYSYSTEM. ;

OUTPUT: PAYSYSTEM-OUTPUTS;
     SYNONYM: PAYOUTS,O1;
     DESCRIPTION;
          THIS OUTPUT REPRESENTS ALL THE REQUIRED OUTPUTS OF THE
          TARGET PAYSYSTEM AS DEFINED BY POLICY. ;

SET: PAYROLL-MASTER-INFORMATION;
     SYNONYM: PAY-MAST,MASTER-FILE,S1;
     DESCRIPTION;
          THIS SET CONTAINS ONE UNIT OF INFORMATION
          FOR EACH EMPLOYEE ON THE PAYROLL, THAT IS,
          THOSE EMPLOYEES WHO ARE TO RECEIVE PAYCHECKS.;

REAL-WORLD-ENTITY: DEPARTMENTS-AND-EMPLOYEES;
     SYNONYM: DEPT-EMP,R1;
     DESCRIPTION;
          THIS IS THE ENTITY WHICH WILL RECEIVE ALL THE OUTPUTS AND
          SUPPLY ALL THE INPUTS. ;

PROCESS: PAYROLL-PROCESSING;
     SYNONYM: PAYPROC,P1;
     DESCRIPTION;
          THIS PROCESS REPRESENTS THE HIGHEST LEVEL PROCESS
          IN THE TARGET SYSTEM.  IT ACCEPTS AND PROCESSES
          ALL INPUTS AND PRODUCES ALL OUTPUTS.;
```

```
/* SYSTEM STRUCTURE   */

INPUT: TIME-CARD;
     PART OF: I1;
     SYNONYM: T-CARD;
     DESCRIPTION;
          THIS INPUT CONTAINS THE INFORMATION ABOUT THE HOURS THAT AN
          HOURLY EMPLOYEE WORKED THE PRECEDING WEEK;

INPUT: HOURLY-EMPLOYMENT-FORM;
     PART OF: EMP-INFO;
     SYNONYM: H-EMP-FORM;
     DESCRIPTION;
          THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
          ADD A NEW HOURLY EMPLOYEE TO THE PAYROLL.;

INPUT: SALARIED-EMPLOYMENT-FORM;
     PART OF: EMP-INFO;
     SYNONYM: S-EMP-FORM;
     DESCRIPTION;
          THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
          ADD A NEW SALARIED EMPLOYEE TO THE PAYROLL.;

INPUT: TAX-WITHHOLDING-CERTIFICATE;
     PART OF: I1;
     SYNONYM: TAX-CERT;
     DESCRIPTION;
         THIS INPUT CONTAINS TAX INFORMATION NECESSARY TO
         COMPUTE THE EMPLOYEE'S PAYCHECK.          ;

INPUT: EMPLOYMENT-TERMINATION-FORM;
     PART OF: EMP-INFO;
     SYNONYM: TERM-FORM;
     DESCRIPTION;
          THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
          DELETE AN EMPLOYEE FROM THE PAYROLL.;

OUTPUT: PAY-STATEMENT;
     PART OF: O1;
     SYNONYM: PAYCHECK;
     DESCRIPTION;
          THIS OUTPUT IS THE PAYMENT TO THE EMPLOYEE FOR THE PREVIOUS
          WORK PERIOD.;

OUTPUT: ERROR-LISTING;
     PART OF: O1;
     SYNONYM: E-LIST;
     DESCRIPTION;
          THIS OUTPUT IS A LISTING OF THAT INPUT DATA THAT FAILED
          THE INPUT VALIDATION RULES.;

OUTPUT: HOURLY-EMPLOYEE-REPORT;
    PART OF: O1;
    SYNONYM: H-EMP-REPORT;
    DESCRIPTION;
         THIS IS AN ADMINISTRATIVE RECORD OF ALL HOURLY
         EMPLOYEES PAID IN ONE WEEK.          ;
```

```
OUTPUT: SALARIED-EMPLOYEE-REPORT;
     PART OF: PAYOUTS;
     SYNONYM: S-EMP-REPORT;
     DESCRIPTION;
          THIS IS AN ADMINISTRATIVE RECORD OF ALL SALARIED
          EMPLOYEES PAID IN ONE MONTH.                    ;

OUTPUT: HIRED-EMPLOYEE-REPORT;
     PART OF: PAYOUTS;
     SYNONYM: HIRED-REPORT;
     DESCRIPTION;
          THIS REPORT PRESENTS INFORMATION ABOUT ALL EMPLOYEES
          HIRED IN THE PREVIOUS WEEK.                     ;

OUTPUT: TERMINATED-EMPLOYEE-REPORT;
     PART OF: PAYOUTS;
     SYNONYM: TERM-REPORT;
     DESCRIPTION;
          THIS REPORT PRESENTS A LISTING OF ALL EMPLOYEES THAT
          ARE NO LONGER ON THE PAYROLL.                   ;

PROCESS PAYPROC;
     SUBPARTS: NEW-EMPLOYEE-PROCESSING,
               TERMINATING-EMP-PROCESSING,
               HOURLY-EMPLOYEE-PROCESSING,
               SALARIED-EMPLOYEE-PROCESSING;

REAL-WORLD-ENTITY: EMPLOYEE;
      PART OF: DEPT-EMP;
      DESCRIPTION;
           AN EMPLOYEE IS IDENTIFIED BY AN EMPLOYEE NUMBER;
      SYNONYM: EMP;

REAL-WORLD-ENTITY PAYROLL-DEPARTMENT;
      PART: DEPT-EMP;
      SYNONYM: PAY-DEPT;
      DESCRIPTION;
           THIS DEPARTMENT IS RESPONSIBLE FOR ALL PAYROLL DATA.;

SET: DEPARTMENT-FILE,HOURLY-EMPLOYEE-FILE,SALARIED-EMPLOYEE-FILE;
      SUBSET: PAY-MAST;
      ROLE: PAY-DEPT;

/* THE FOLLOWING STATEMENTS PRESENT SYSTEM FLOW WITH RESPECT TO
   THE OBJECT DEFINED PREVIOUSLY.                          */

REAL-WORLD-ENTITY: EMP;
     RECEIVES: PAY-STATEMENT;
     GENERATES: TAX-WITHHOLDING-CERTIFICATE,T-CARD;

REAL-WORLD-ENTITY: PAY-DEPT;
     RECEIVES: E-LIST,HIRED-REPORT,TERM-REPORT,
               SALARIED-EMPLOYEE-REPORT,
               HOURLY-EMPLOYEE-REPORT;
     GENERATES: HOURLY-EMPLOYMENT-FORM,
               SALARIED-EMPLOYMENT-FORM,
               EMPLOYMENT-TERMINATION-FORM;
     RESPONSIBLE FOR PAY-MAST;
```

```
PROCESS: NEW-EMPLOYEE-PROCESSING;
     RECEIVES: HOURLY-EMPLOYMENT-FORM,
               SALARIED-EMPLOYMENT-FORM,
               TAX-WITHHOLDING-CERTIFICATE;
     GENERATES: HIRED-REPORT;
     DESCRIPTION;
          THIS PROCESS STORES INFORMATION ABOUT NEW EMPLOYEES AND
          THEN PRINTS OUT A CORRESPONDING REPORT.          ;

PROCESS: TERMINATING-EMP-PROCESSING;
     RECEIVES: EMPLOYMENT-TERMINATION-FORM;
     GENERATES: TERM-REPORT;
     DESCRIPTION;
          THIS PROCESS DELETES DATA, FOR THOSE EMPLOYEES WHO
          ARE NO LONGER ON THE PAYROLL, FROM THE FILES. IT ALSO
          PRINTS A LIST OF ALL EMPLOYEES NO LONGER ON THE
          PAYROLL.                                         ;

PROCESS: HOURLY-EMPLOYEE-PROCESSING;
     RECEIVES: T-CARD;
     GENERATES: PAYCHECK,ERROR-LISTING,HOURLY-EMPLOYEE-REPORT;
     DESCRIPTION;
          THIS PROCESS PERFORMS THOSE ACTIONS NEEDED TO INTERPRET
          TIME CARDS TO PRODUCE A PAY STATEMENT FOR EACH HOURLY
          EMPLOYEE.                                  ;

PROCESS: SALARIED-EMPLOYEE-PROCESSING;
     GENERATES: PAYCHECK, ERROR-LISTING,
                SALARIED-EMPLOYEE-REPORT;
     DESCRIPTION;
          THIS PROCESS PRODUCES THE PAY STATEMENT FOR SALARIED
          EMPLOYEES ONCE A MONTH.                    ;

EOF
```

PSA VERSION 2.1R5 1977.138

BELLST IDT/GFK KARLSRUHE

FORMATTED PROBLEM STATEMENT

PARAMETERS FOR: FPS

```
FILE NOINDEX PRINT NOPUNCH SMARG=5 NMARG=20 AMARG=10 BMARG=25 RNMARG=70 CM.
ONE-PER-LINE DEFINE COMMENT NONEW-PAGE NONEW-LINE

 1 SET                                          DEPARTMENT-FILE;
 2     SUBSET OF:      PAYROLL-MASTER-INFORMATION;
 3     RESPONSIBLE-INTERFACE IS:
 4                     PAYROLL-DEPARTMENT;
 5
 6 INTERFACE                                    DEPARTMENTS-AND-EMPLOYEES;
 7     SYNONYMS ARE:   DEPT-EMP,
 8                     R1;
 9     DESCRIPTION;
10           THIS IS THE ENTITY WHICH WILL RECEIVE ALL THE OUTPUTS AND
11           SUPPLY ALL THE INPUTS.;
12     SUBPARTS ARE:   EMPLOYEE,
13                     PAYROLL-DEPARTMENT;
14     GENERATES:      EMPLOYEE-INFORMATION;
15     RECEIVES:       PAYSYSTEM-OUTPUTS;
16
17 DESIGNATE                                   DEPT-EMP
18      AS A SYNONYM FOR DEPARTMENTS-AND-EMPLOYEES;
19
20 DESIGNATE                                   E-LIST
21      AS A SYNONYM FOR ERROR-LISTING;
22
23 DESIGNATE                                   EMP
24      AS A SYNONYM FOR EMPLOYEE;
25
26 DESIGNATE                                   EMP-INFO
27      AS A SYNONYM FOR EMPLOYEE-INFORMATION;
28
29 INTERFACE                                   EMPLOYEE;
30     SYNONYMS ARE:   EMP;
31     DESCRIPTION;
32           AN EMPLOYEE IS IDENTIFIED BY AN EMPLOYEE NUMBER;
33     PART OF:        DEPARTMENTS-AND-EMPLOYEES;
34     GENERATES:      TAX-WITHHOLDING-CERTIFICATE,
35                     TIME-CARD;
36     RECEIVES:       PAY-STATEMENT;
37
38 INPUT                                       EMPLOYEE-INFORMATION;
39     SYNONYMS ARE:   EMP-INFO,
40                     I1;
41     DESCRIPTION;
42           THIS INPUT REPRESENTS ALL THE NECESSARY INFORMATION TO
43           PRODUCE THE OUTPUTS FROM THE PAYSYSTEM.;
44     SUBPARTS ARE:   TIME-CARD,
45                     HOURLY-EMPLOYMENT-FORM,
46                     SALARIED-EMPLOYMENT-FORM,
47                     TAX-WITHHOLDING-CERTIFICATE,
48                     EMPLOYMENT-TERMINATION-FORM;
49     GENERATED BY:   DEPARTMENTS-AND-EMPLOYEES;
50     RECEIVED BY:    PAYROLL-PROCESSING;
```

PSA VERSION 2.1R5 1977.133

BELLST IDT/GFK KARLSRUHE

FORMATTED PROBLEM STATEMENT

```
51
52 INPUT                                          EMPLOYMENT-TERMINATION-FORM;
53     SYNONYMS ARE:  TERM-FORM;
54     DESCRIPTION;
55           THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
56           DELETE AN EMPLOYEE FROM THE PAYROLL.;
57     PART OF:       EMPLOYEE-INFORMATION;
58     GENERATED BY:  PAYROLL-DEPARTMENT;
59     RECEIVED BY:   TERMINATING-EMP-PROCESSING;
60
61 OUTPUT                                         ERROR-LISTING;
62     SYNONYMS ARE:  E-LIST;
63     DESCRIPTION;
64           THIS OUTPUT IS A LISTING OF THAT INPUT DATA THAT FAILED
65           THE INPUT VALIDATION RULES.;
66     PART OF:       PAYSYSTEM-OUTPUTS;
67     GENERATED BY:  HOURLY-EMPLOYEE-PROCESSING,
68                    SALARIED-EMPLOYEE-PROCESSING;
69     RECEIVED BY:   PAYROLL-DEPARTMENT;
70
71 DESIGNATE                                      H-EMP-FORM
72      AS A SYNONYM FOR HOURLY-EMPLOYMENT-FORM;
73
74 DESIGNATE                                      H-EMP-REPORT
75      AS A SYNONYM FOR HOURLY-EMPLOYEE-REPORT;
76
77 OUTPUT                                         HIRED-EMPLOYEE-REPORT;
78     SYNONYMS ARE:  HIRED-REPORT;
79     DESCRIPTION;
80          THIS REPORT PRESENTS INFORMATION ABOUT ALL EMPLOYEES
81          HIRED IN THE PREVIOUS WEEK.;
82     PART OF:       PAYSYSTEM-OUTPUTS;
83     GENERATED BY:  NEW-EMPLOYEE-PROCESSING;
84     RECEIVED BY:   PAYROLL-DEPARTMENT;
85
86 DESIGNATE                                      HIRED-REPORT
87      AS A SYNONYM FOR HIRED-EMPLOYEE-REPORT;
88
89 SET                                           HOURLY-EMPLOYEE-FILE;
90     SUBSET OF:     PAYROLL-MASTER-INFORMATION;
91     RESPONSIBLE-INTERFACE IS:
92                    PAYROLL-DEPARTMENT;
93
94 PROCESS                                       HOURLY-EMPLOYEE-PROCESSING;
95     DESCRIPTION;
96          THIS PROCESS PERFORMS THOSE ACTIONS NEEDED TO INTERPRET
97          TIME CARDS TO PRODUCE A PAY STATEMENT FOR EACH HOURLY
98          EMPLOYEE.;
99     PART OF:       PAYROLL-PROCESSING;
100    RECEIVES:      TIME-CARD;
101    GENERATES:     PAY-STATEMENT,
102                   ERROR-LISTING,
103                   HOURLY-EMPLOYEE-REPORT;
104
105 OUTPUT                                        HOURLY-EMPLOYEE-REPORT;
```

FORMATTED PROBLEM STATEMENT

```
106      SYNONYMS ARE:  H-EMP-REPORT;
107      DESCRIPTION;
108           THIS IS AN ADMINISTRATIVE RECORD OF ALL HOURLY
109           EMPLOYEES PAID IN ONE WEEK.;
110      PART OF:       PAYSYSTEM-OUTPUTS;
111      GENERATED BY:  HOURLY-EMPLOYEE-PROCESSING;
112      RECEIVED BY:   PAYROLL-DEPARTMENT;
113
114 INPUT                                   HOURLY-EMPLOYMENT-FORM;
115      SYNONYMS ARE:  H-EMP-FORM;
116      DESCRIPTION;
117           THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
118           ADD A NEW HOURLY EMPLOYEE TO THE PAYROLL.;
119      PART OF:       EMPLOYEE-INFORMATION;
120      GENERATED BY:  PAYROLL-DEPARTMENT;
121      RECEIVED BY:   NEW-EMPLOYEE-PROCESSING;
122
123 DESIGNATE                               I1
124       AS A SYNONYM FOR EMPLOYEE-INFORMATION;
125
126 DESIGNATE                               MASTER-FILE
127       AS A SYNONYM FOR PAYROLL-MASTER-INFORMATION;
128
129 PROCESS                                 NEW-EMPLOYEE-PROCESSING;
130      DESCRIPTION;
131           THIS PROCESS STORES INFORMATION ABOUT NEW EMPLOYEES AND
132           THEN PRINTS OUT A CORRESPONDING REPORT.;
133      PART OF:       PAYROLL-PROCESSING;
134      RECEIVES:      HOURLY-EMPLOYMENT-FORM,
135                     SALARIED-EMPLOYMENT-FORM,
136                     TAX-WITHHOLDING-CERTIFICATE;
137      GENERATES:     HIRED-EMPLOYEE-REPORT;
138
139 DESIGNATE                               O1
140       AS A SYNONYM FOR PAYSYSTEM-OUTPUTS;
141
142 DESIGNATE                               PAY-DEPT
143       AS A SYNONYM FOR PAYROLL-DEPARTMENT;
144
145 DESIGNATE                               PAY-MAST
146       AS A SYNONYM FOR PAYROLL-MASTER-INFORMATION;
147
148 OUTPUT                                  PAY-STATEMENT;
149      SYNONYMS ARE:  PAYCHECK;
150      DESCRIPTION;
151           THIS OUTPUT IS THE PAYMENT TO THE EMPLOYEE FOR THE PREVIOUS
152           WORK PERIOD.;
153      PART OF:       PAYSYSTEM-OUTPUTS;
154      GENERATED BY:  HOURLY-EMPLOYEE-PROCESSING,
155                     SALARIED-EMPLOYEE-PROCESSING;
156      RECEIVED BY:   EMPLOYEE;
157
158 DESIGNATE                               PAYCHECK
159       AS A SYNONYM FOR PAY-STATEMENT;
160
```

FORMATTED PROBLEM STATEMENT

```
161 DESIGNATE                                   PAYOUTS
162      AS A SYNONYM FOR PAYSYSTEM-OUTPUTS;
163
164 DESIGNATE                                   PAYPROC
165      AS A SYNONYM FOR PAYROLL-PROCESSING;
166
167 INTERFACE                                   PAYROLL-DEPARTMENT;
168      SYNONYMS ARE:  PAY-DEPT;
169      DESCRIPTION;
170            THIS DEPARTMENT IS RESPONSIBLE FOR ALL PAYROLL DATA.;
171      PART OF:       DEPARTMENTS-AND-EMPLOYEES;
172      GENERATES:     HOURLY-EMPLOYMENT-FORM,
173                     SALARIED-EMPLOYMENT-FORM,
174                     EMPLOYMENT-TERMINATION-FORM;
175      RECEIVES:      ERROR-LISTING,
176                     HIRED-EMPLOYEE-REPORT,
177                     TERMINATED-EMPLOYEE-REPORT,
178                     SALARIED-EMPLOYEE-REPORT,
179                     HOURLY-EMPLOYEE-REPORT;
180      RESPONSIBLE FOR:
181                    DEPARTMENT-FILE,
182                    HOURLY-EMPLOYEE-FILE,
183                    SALARIED-EMPLOYEE-FILE,
184                    PAYROLL-MASTER-INFORMATION;
185
186 SET                                         PAYROLL-MASTER-INFORMATION;
187      SYNONYMS ARE:  MASTER-FILE,
188                     PAY-MAST,
189                     S1;
190      DESCRIPTION;
191            THIS SET CONTAINS ONE UNIT OF INFORMATION
192            FOR EACH EMPLOYEE ON THE PAYROLL, THAT IS,
193            THOSE EMPLOYEES WHO ARE TO RECEIVE PAYCHECKS.;
194      SUBSETS ARE:   DEPARTMENT-FILE,
195                     HOURLY-EMPLOYEE-FILE,
196                     SALARIED-EMPLOYEE-FILE;
197      UPDATED BY:    PAYROLL-PROCESSING;
198      RESPONSIBLE-INTERFACE IS:
199                     PAYROLL-DEPARTMENT;
200
201 PROCESS                                     PAYROLL-PROCESSING;
202      SYNONYMS ARE:  PAYPROC,
203                     P1;
204      DESCRIPTION;
205            THIS PROCESS REPRESENTS THE HIGHEST LEVEL PROCESS
206            IN THE TARGET SYSTEM.  IT ACCEPTS AND PROCESSES
207            ALL INPUTS AND PRODUCES ALL OUTPUTS.;
208      SUBPARTS ARE:  NEW-EMPLOYEE-PROCESSING,
209                     TERMINATING-EMP-PROCESSING,
210                     HOURLY-EMPLOYEE-PROCESSING,
211                     SALARIED-EMPLOYEE-PROCESSING;
212      RECEIVES:      EMPLOYEE-INFORMATION;
213      GENERATES:     PAYSYSTEM-OUTPUTS;
214      UPDATES:       PAYROLL-MASTER-INFORMATION;
215
```

FORMATTED PROBLEM STATEMENT

```
OUTPUT                                          PAYSYSTEM-OUTPUTS;
     SYNONYMS ARE:  01,
                    PAYOUTS;
     DESCRIPTION;
          THIS OUTPUT REPRESENTS ALL THE REQUIRED OUTPUTS OF THE
          TARGET PAYSYSTEM AS DEFINED BY POLICY.;
     SUBPARTS ARE:  PAY-STATEMENT,
                    ERROR-LISTING,
                    HOURLY-EMPLOYEE-REPORT,
                    SALARIED-EMPLOYEE-REPORT,
                    HIRED-EMPLOYEE-REPORT,
                    TERMINATED-EMPLOYEE-REPORT;
     GENERATED BY:  PAYROLL-PROCESSING;
     RECEIVED BY:   DEPARTMENTS-AND-EMPLOYEES;

DESIGNATE                                       P1
      AS A SYNONYM FOR PAYROLL-PROCESSING;

DESIGNATE                                       R1
      AS A SYNONYM FOR DEPARTMENTS-AND-EMPLOYEES;

DESIGNATE                                       S-EMP-FORM
      AS A SYNONYM FOR SALARIED-EMPLOYMENT-FORM;

DESIGNATE                                       S-EMP-REPORT
      AS A SYNONYM FOR SALARIED-EMPLOYEE-REPORT;

SET                                             SALARIED-EMPLOYEE-FILE;
     SUBSET OF:      PAYROLL-MASTER-INFORMATION;
     RESPONSIBLE-INTERFACE IS:
                     PAYROLL-DEPARTMENT;

PROCESS                                         SALARIED-EMPLOYEE-PROCESSING;
     DESCRIPTION;
          THIS PROCESS PRODUCES THE PAY STATEMENT FOR SALARIED
          EMPLOYEES ONCE A MONTH.;
     PART OF:        PAYROLL-PROCESSING;
     GENERATES:      PAY-STATEMENT,
                     ERROR-LISTING,
                     SALARIED-EMPLOYEE-REPORT;

OUTPUT                                          SALARIED-EMPLOYEE-REPORT;
     SYNONYMS ARE:  S-EMP-REPORT;
     DESCRIPTION;
          THIS IS AN ADMINISTRATIVE RECORD OF ALL SALARIED
          EMPLOYEES PAID IN ONE MONTH.;
     PART OF:        PAYSYSTEM-OUTPUTS;
     GENERATED BY:   SALARIED-EMPLOYEE-PROCESSING;
     RECEIVED BY:    PAYROLL-DEPARTMENT;

INPUT                                           SALARIED-EMPLOYMENT-FORM;
     SYNONYMS ARE:  S-EMP-FORM;
     DESCRIPTION;
          THIS INPUT CONTAINS THE INFORMATION NECESSARY TO
          ADD A NEW SALARIED EMPLOYEE TO THE PAYROLL.;
```

FORMATTED PROBLEM STATEMENT

```
271      PART OF:        EMPLOYEE-INFORMATION;
272      GENERATED BY:   PAYROLL-DEPARTMENT;
273      RECEIVED BY:    NEW-EMPLOYEE-PROCESSING;
274
275  DESIGNATE                                   S1
276       AS A SYNONYM FOR PAYROLL-MASTER-INFORMATION;
277
278  DESIGNATE                                   T-CARD
279       AS A SYNONYM FOR TIME-CARD;
280
281  DESIGNATE                                   TAX-CERT
282       AS A SYNONYM FOR TAX-WITHHOLDING-CERTIFICATE;
283
284  INPUT                                       TAX-WITHHOLDING-CERTIFICATE;
285      SYNONYMS ARE:   TAX-CERT;
286      DESCRIPTION;
287           THIS INPUT CONTAINS TAX INFORMATION NECESSARY TO
288           COMPUTE THE EMPLOYEE'S PAYCHECK.;
289      PART OF:        EMPLOYEE-INFORMATION;
290      GENERATED BY:   EMPLOYEE;
291      RECEIVED BY:    NEW-EMPLOYEE-PROCESSING;
292
293  DESIGNATE                                   TERM-FORM
294       AS A SYNONYM FOR EMPLOYMENT-TERMINATION-FORM;
295
296  DESIGNATE                                   TERM-REPORT
297       AS A SYNONYM FOR TERMINATED-EMPLOYEE-REPORT;
298
299  OUTPUT                                      TERMINATED-EMPLOYEE-REPORT;
300      SYNONYMS ARE:   TERM-REPORT;
301      DESCRIPTION;
302           THIS REPORT PRESENTS A LISTING OF ALL EMPLOYEES THAT
303           ARE NO LONGER ON THE PAYROLL.;
304      PART OF:        PAYSYSTEM-OUTPUTS;
305      GENERATED BY:   TERMINATING-EMP-PROCESSING;
306      RECEIVED BY:    PAYROLL-DEPARTMENT;
307
308  PROCESS                                     TERMINATING-EMP-PROCESSING;
309      DESCRIPTION;
310           THIS PROCESS DELETES DATA, FOR THOSE EMPLOYEES WHO
311           ARE NO LONGER ON THE PAYROLL, FROM THE FILES. IT ALSO
312           PRINTS A LIST OF ALL EMPLOYEES NO LONGER ON THE
313           PAYROLL.;
314      PART OF:        PAYROLL-PROCESSING;
315      RECEIVES:       EMPLOYMENT-TERMINATION-FORM;
316      GENERATES:      TERMINATED-EMPLOYEE-REPORT;
317
318  INPUT                                       TIME-CARD;
319      SYNONYMS ARE:   T-CARD;
320      DESCRIPTION;
321            THIS INPUT CONTAINS THE INFORMATION ABOUT THE HOURS THAT AN
322            HOURLY EMPLOYEE WORKED THE PRECEDING WEEK;
323      PART OF:        EMPLOYEE-INFORMATION;
324      GENERATED BY:   EMPLOYEE;
325      RECEIVED BY:    HOURLY-EMPLOYEE-PROCESSING;
```

BELUST IDT/GFK KARLSRUHE

NAME LIST

PARAMETERS FOR: NL

ORDER=BYTYPE

	NAME	TYPE	SYNONYM	
1	ARRIVAL-TYPE	ATTRIBUTE		-
2	COMPLEXITY-LEVEL	ATTRIBUTE		-
3	COPIES	ATTRIBUTE		-
4	DATA-STANDARD	ATTRIBUTE		-
5	TYPE	ATTRIBUTE		-
6	CHARACTER	ATTRIBUTE-VALUE		
7	DATE	ATTRIBUTE-VALUE		
8	HIGH	ATTRIBUTE-VALUE		
9	LOW	ATTRIBUTE-VALUE		-
10	MEDIUM	ATTRIBUTE-VALUE		-
11	NUMERIC		DEPT-FILE	
12	RANDOM		E-EMP-FILE	
13	SCHEDULED	SET	MASTER-FILE	
			PAY-MAST	
			S1	
	-SALARIED-EMPLOYEE-FILE	SET	S-EMP-FILE	
170	CONSTANTINE	SOURCE		-
171	MANY	SYSTEM-PARAMETER		-
172	NO-OF-DEPARTMENTS	SYSTEM-PARAMETER		-
173	NO-OF-EMPLOYEES	SYSTEM-PARAMETER		-
174	NO-OF-HOURLY-EMPLOYEES	SYSTEM-PARAMETER		-
175	NO-OF-PAYROLL-PROCESSING	SYSTEM-PARAMETER		-
176	NO-OF-SALARIED-EMPLOYEES	SYSTEM-PARAMETER		-
177	ONE	SYSTEM-PARAMETER		-
178	SEVERAL	SYSTEM-PARAMETER		-

Kommando : NAME-LIST ORDER = BYTYPE
(Anfang und Ende)

PSA VERSION 2.1R5 1977.122

BELUST IDT/GFK KARLSRUHE

PROCESS STRUCTURE

PARAMETERS FOR: STR

PROCESS INDENT=3 NOINDEX

```
COUNT LEVEL NAME

    1     1 PAYROLL-PROCESSING
    2      2   NEW-EMPLOYEE-PROCESSING
    3       3     SALARIED-RECORD-CREATION
    4       3     HOURLY-RECORD-CREATION
    5       3     HIRE-REPORT-ENTRY-GENERATION
    6       3     DEPARTMENT-FILE-ADDITION
    7      2   TERMINATING-EMP-PROCESSING
    8       3     SALARIED-RECORD-DELETION
    9       3     HOURLY-RECORD-DELETION
   10       3     TERM-REPORT-ENTRY-GENERATION
   11       3     DEPARTMENT-FILE-REMOVAL
   12      2   HOURLY-EMPLOYEE-PROCESSING
   13       3     HOURLY-PAYCHECK-VALIDATION
   14        4       TIME-CARD-VALIDATION
   15       3     HOURLY-EMP-UPDATE
   16        4       HOURS-UPDATE
   17       3     H-REPORT-ENTRY-GENERATION
   18       3     HOURLY-PAYCHECK-PRODUCTION
   19        4       H-GROSS-PAY-COMPUTATION
   20        4       TOTAL-HOURS-COMPUTATION
   21      2   SALARIED-EMPLOYEE-PROCESSING
   22       3     SALARIED-PAYCHECK-VALIDATION
   23       3     SALARIED-EMP-UPDATE
   24       3     S-REPORT-ENTRY-GENERATION
   25       3     SALARIED-PAYCHECK-PRODUCTION
   26        4       S-GROSS-PAY-COMPUTATION
   27      2   SHARED-ROUTINES
   28       3     PAY-COMPUTATION-VALIDATION
   29       3     TAX-COMPUTATION
   30       3     NET-PAY-COMPUTATION
   31       3     TOTAL-DEDUCTIONS-COMPUTATION
   32       3     GROSS-PAY-UPDATE
   33       3     FEDERAL-DEDUCTIONS-UPDATE
   34       3     STATE-DEDUCTIONS-UPDATE
   35       3     FICA-DEDUCTIONS-UPDATE
   36       3     FUNDS-UPDATE
```

LEVEL	COUNT	LEVEL	COUNT	LEVEL	COUNT	LEVEL	COUNT	LEVEL	COUNT
1	1	2	5	3	25	4	5		

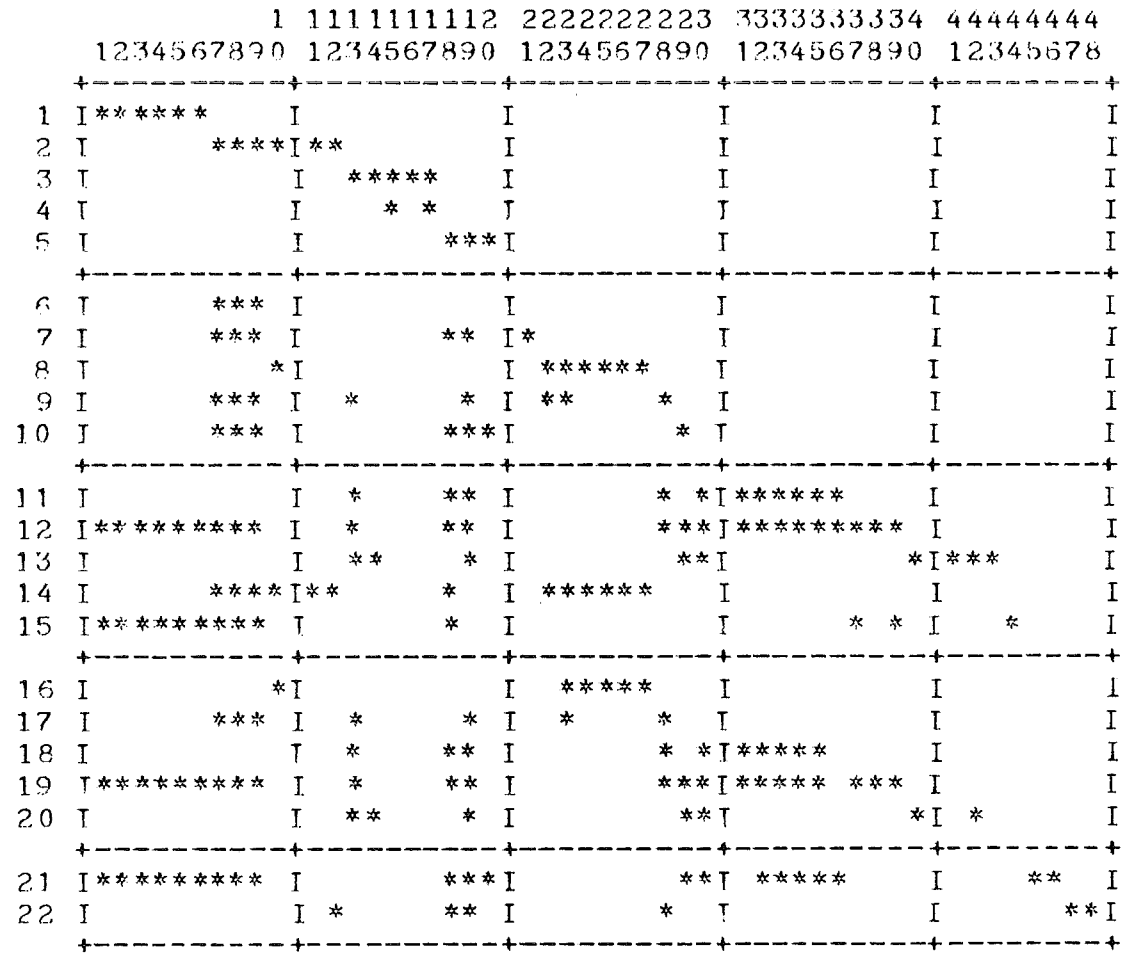
PSA VERSION 2.105 1977.122

BELUST IDT/GFK KARLSRUHE

CONSISTS COMPARISON REPORT

BASIC CONTENTS MATRIX

An * in (i,j) means that column j is contained directly or indirectly in row i. The columns do not consist of anything further. Intermediate groups are ignored.



BELUST IDT/GFK KARLSRUHE

CONSISTS COMPARISON REPORT

CONTENTS SIMILARITY MATRIX

The number in (i,i) is the number of objects
at the lowest level contained in row i from above.

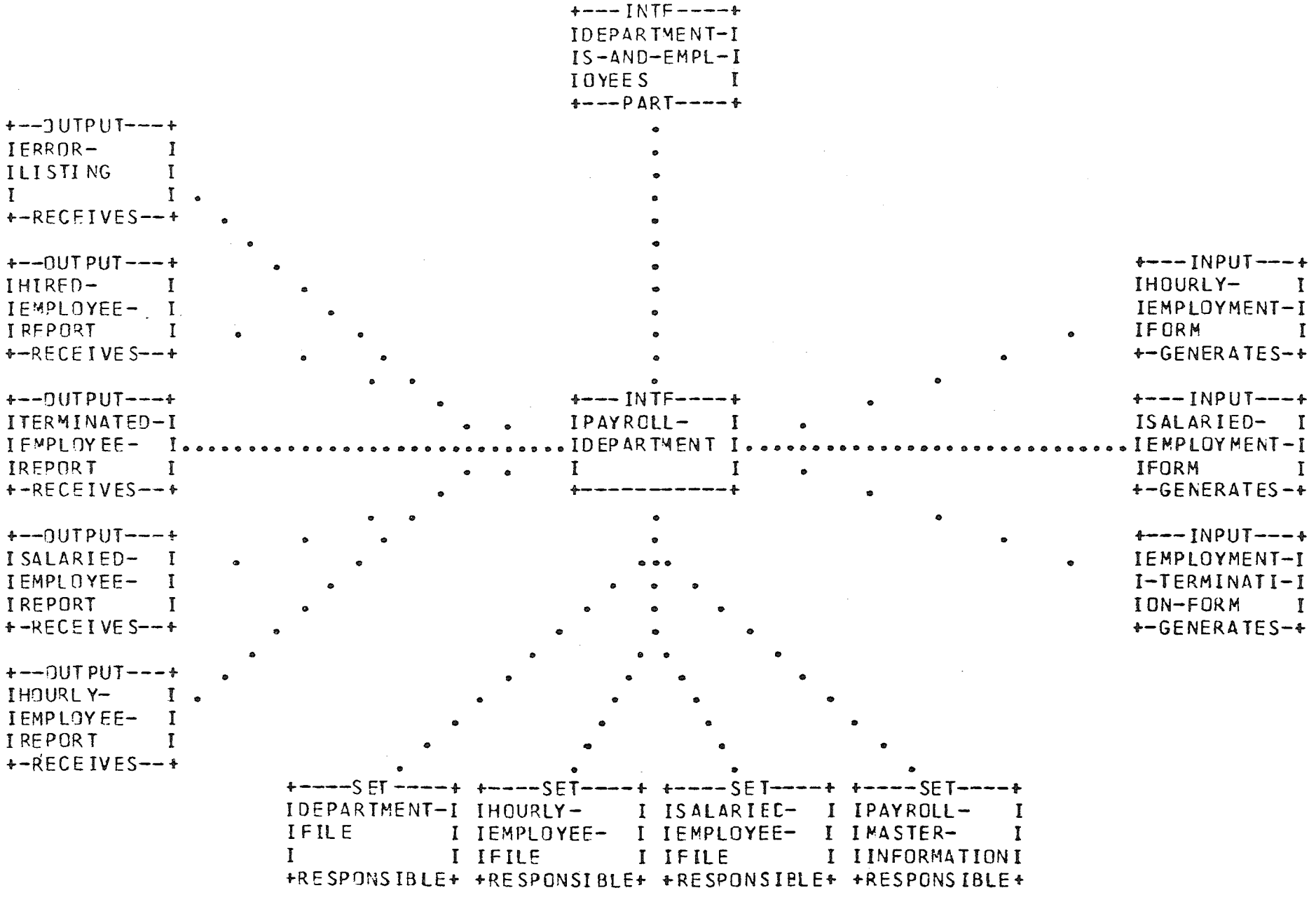
The number in (i,j) (i not equal j) is
the number of objects at the lowest level in
common between rows i and j from above.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
1	6											6			6				6		6	
2		6				3	3	1	3	3		3		6	3	1	3		3		3	1
3			5	2					1		1	1	2				1	1	1	2		
4				2																		
5					3		2		1	3	2	2	1	1	1		1	2	2	1	3	2
6						3	3		3	3		3		3	3		3		3		3	
7							6		4	5	2	5	1	4	4		4	2	5	1	5	2
8								7	2					7		6	1					
9									8	4	3	6	2	5	3	1	7	3	6	2	4	2
10										7	2	6	2	4	4		4	2	6	2	7	2
11											11	11	3	1	1		3	10	10	3	8	3
12												24	4	4	12		6	10	23	4	18	3
13													9				2	3	4	7	3	1
14														13	4	6	4	1	4		4	2
15															13		3	1	12		10	1
16																6	1					
17																	7	3	6	2	4	2
18																		10	10	3	7	3
19																			23	4	17	3
20																				7	3	1
21																					21	2
22																						6

BELUST IDT/GFK KARLSRUHE

INTERFACE PICTURE

PAYROLL-DEPARTMENT



4.3 SREP - Software Requirements Engineering Program

/Al 76/, /BeBi 76/, /DaVi 76/

4.3.1 Ziele des SREP

Entwicklung von Methoden und Hilfsmitteln zur Senkung der Entwicklungskosten und -zeiten. Dies sollte vor allem durch Formalisierung der Spezifikations- und Entwurfsphase erreicht werden, um die hier entstehenden Fehler frühzeitig in den Griff zu bekommen.

- Entwicklung einer formalen Sprache zur Beschreibung der Anforderungen, die die folgenden Eigenschaften unterstützen sollte: Eindeutigkeit, Entwurfsunabhängigkeit, Testbarkeit, Modularität und bessere Kommunikation.

- Entwicklung rechnergestützter Hilfsmittel zur Prüfung auf Widerspruchsfreiheit, Vollständigkeit und Korrektheit.

Aufgrund des speziellen Anwendungsgebiets von SREP - große Realzeitsysteme - werden zwei zusätzliche Forderungen aufgestellt:

- Leistungs- und Zeitanforderungen sollten in überprüfbarer Form ausgedrückt werden können

- Die dynamischen Eigenschaften des Systems sollten durch Simulation überprüfbar sein.

Bemerkung: In PSL kann zwar das Zeitverhalten eines Systems durch EVENTs, die andere CONDITIONS und EVENTs-triggern, ausgedrückt werden, der Analysator ist jedoch derzeit noch nicht in der Lage, diese Zeitbedingungen zu korrelieren oder eine Folge von widersprüchlichen oder unmöglichen Zeitanforderungen zu erkennen.

4.3.2 Beschreibungsmodell von SREP

SREP basiert in seiner Philosophie auf PSL/PSA und erlaubt, abgesehen von einer unterschiedlichen Terminologie, die Beschreibung der gleichen Systemaspekte.

Da in Realzeitsystemen die Kommunikation zwischen Moduln und deren zeitlicher Ablauf eine wesentliche Rolle spielen, enthält das Beschreibungsmodell von SREP entscheidende Erweiterungen bei der Beschreibung des dynamischen Verhaltens von Systemen.

Diese Erweiterungen lassen sich in zwei Systemaspekten zusammenfassen:

- Beschreibung der Anforderungen an den Ablauf von Prozessen
- Formulierung von überprüfbaren Leistungsanforderungen (performance)

Auf die durch diese Erweiterung möglich gewordenen dynamischen Analysen (Simulation) wird im Abschnitt Prüfmittel eingegangen.

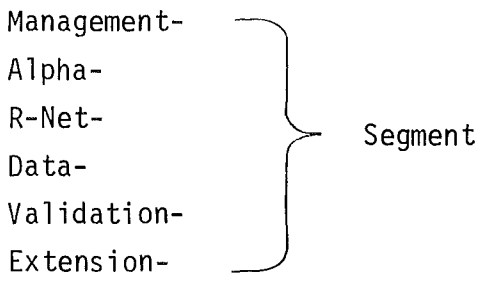
Die syntaktische Realisierung des Beschreibungsmodells wird durch die Spezifikationssprache RSL definiert. Objekte, Attribute und Relationen sind aus PSL nach RSL übernommen. Hinzugefügt wurde die Darstellung des Realzeitsystems durch die sogenannten R-Nets (Requirement-Nets), die die Reaktionen des Systems auf äußere Ereignisse beschreiben (stimulus-response-Darstellung). In RSL wird dieses Ablaufmodell durch Erweiterung des schon in PSL eingeführten Strukturkonzepts (dort nur statisch) beschrieben.

Es gibt zwei Typen von Strukturen:

- R-Net (SUBNET) : es modelliert den Ablauf der Verarbeitungsschritte und dient zur Spezifikation der Systemantworten auf verschiedene Ereignisse.

- Validation-Path : ein Graph, dessen Knoten Daten zum Test von Leistungsanforderungen enthält und dessen Kanten mit den möglichen Wegen in der entsprechenden R-Net/Subnet Struktur übereinstimmen. (In der graph. Repräsentation werden die Knoten deshalb direkt in das R-Net eingezeichnet).

Eine RSL-Beschreibung besteht aus einer oder mehreren Abschnitten, die "definitions" genannt werden. Eine "definition" besteht wiederum aus Segmenten, denen man die verschiedenen Systemaspekte zuordnen kann:



- Management-Segment

Dieses Segment beschreibt Informationen die für das Projektmanagement von Bedeutung sind:

- Beschreibungsänderungen
- Referenzen zu Hintergrundmaterial
- Synonyme, Kommentare

- Alpha-Segment

Dieses Segment beschreibt die grundlegenden Verarbeitungsschritte, die im System enthalten sein sollen (entspricht etwa einem

PROCESS in PSL). Innerhalb eines Alpha können der funktionalen Beschreibung der Anforderungen ausführbare Beschreibungen (zu Simulationszwecken) durch die Relationen BETA und GAMMA zugeordnet werden. Die BETAs sind funktionelle Modelle eines ALPHA und dienen der funktionellen Simulation. Die GAMMAs sind analytische Modelle von Algorithmen eines analytischen Tests und dienen der analytischen Emulation. Beziehungen zu den Daten des Datensegments werden durch spezielle Sprachelemente von RSL ausgedrückt: SELECT, CREATE, DESTROY, FORM, FOR EACH.

Beispiel:

```
ALPHA: find-student-address.
    DESCRIPTION: This Alpha finds the address
                 of the student given a specified
                 student-id.
    COMPLETENESS: INCOMPLETE.
    INPUTS      : student-id.
    OUTPUTS     : studadr.
    BETA        :
                  SELECT NEXT FROM student-file
                  SUCH THAT student-id = student-file.student-id
END;
```

Im Datensegment existieren die folgenden Datenbeschreibungen:

```
FILE: student-file.
      CONTAINS DATA: student-id, student-name.
      ORDERED BY   : student-id.
      LOCALITY     : GLOBAL
      INPUT TO     : find-student-address,
                     get student-information.
```

```
DATA: student-id.
  TYPE: INTEGER
  LOCALITY: LOCAL
  MINIMUM VALUE: 0.
  INPUT TO: find-student-
            address
```

R-Net-Segment

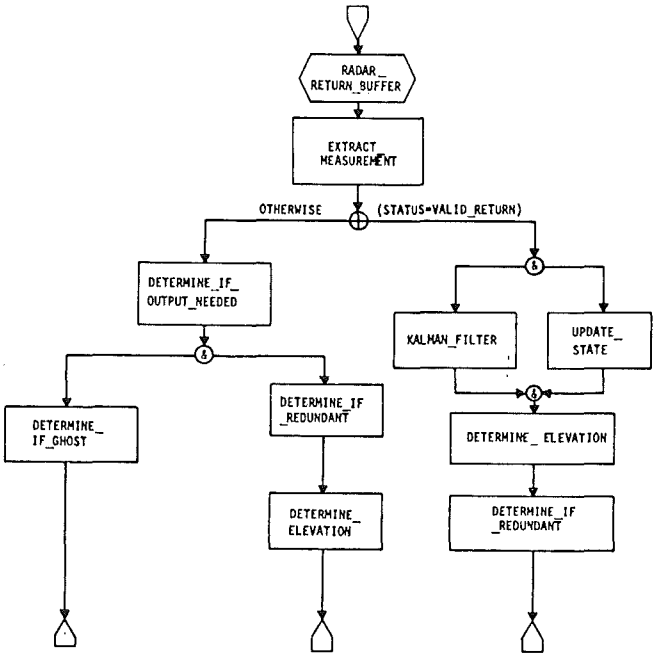
Ein R-Net spezifiziert die Folge von ALPHAs, die erforderlich ist, um die gewünschten Zustandsänderungen durchzuführen und die entsprechenden Systemantworten zu erzeugen. Verschiedene Beschreibungsebenen können durch SUBNETS spezifiziert werden (hierarchische Graphenstruktur).

Ein R-Net besteht aus einer Menge von Knoten, die ALPHAs und SUBNETs repräsentieren, und aus einer Menge von Strukturierungsknoten, die die möglichen Ablaufstrukturen charakterisieren. Es gibt drei Arten von Strukturierungs-Knoten: AND, OR und FOR EACH. AND-Knoten dienen zur Beschreibung von parallelen Ablauffolgen (mit impliziter Synchronisation am Ende der Verzweigung), OR-Knoten zur Beschreibung bedingter Ausführungsfolgen; FOR EACH gibt an, daß ein Prozess oder eine Menge von Prozessen wiederholt auszuführen ist.

Beispiel 1:

```
R_NET:  PROCESS_RADAR_RETURN.
    STRUCTURE:
        INPUT_INTERFACE RADAR_RETURN_BUFFER
        EXTRACT_MEASUREMENT
        DO (STATUS = VALID_RETURN)
            DO  UPDATE_STATE AND KALMAN_FILTER  END
            DETERMINE_ELEVATION
            DETERMINE_IF_REDUNDANT
            TERMINATE
        OTHERWISE
            DETERMINE_IF_OUTPUT_NEEDED
            DO  DETERMINE_IF_REDUNDANT
                DETERMINE_ELEVATION
                TERMINATE
            AND DETERMINE_IF_GHOST
                TERMINATE
            END
        END
    END.
```

Sample R_NET in RSL.

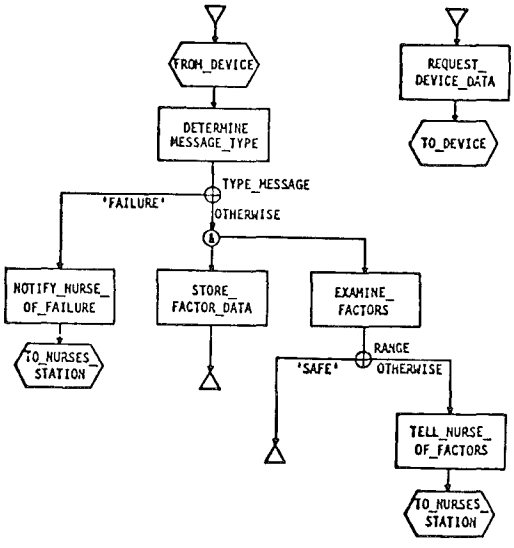


Flow graph of a sample R_NET.

Beispiel 2: Krankenhausüberwachung /Al 76/

The problem is the following:

"(1) A patient monitoring program is required for a hospital. (2) Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood-pressure, and skin resistance. (3) The program reads these factors on a periodic basis (specified for each patient) and stores these factors in a data base. (4) For each patient, safe ranges for each factor are specified (e.g., patient X's valid temperature range is 98 to 99.5 degrees Fahrenheit). (5) If a factor falls outside of the patient's safe range, or if an analog device fails, the nurse's station is notified".



R-NET

```
ORIGINATING_REQUIREMENT: SENTENCE_2.
    DESCRIPTION: "DEFINES ANALOG DEVICE MEASUREMENTS".
    TRACES TO: MESSAGE DEVICE_REPORT.

MESSAGE: DEVICE_REPORT.
    PASSED THROUGH: INPUT_INTERFACE FROM_DEVICE.
    MADE BY: DATA DEVICE_NUMBER, DATA TYPE_MESSAGE,
             DATA DEVICE_DATA.
    TRACED FROM: SENTENCE_2.

DATA: DEVICE_DATA.
    INCLUDES: DATA PULSE, DATA TEMPERATURE,
              DATA BLOOD_PRESSURE,
              DATA SKIN_RESISTANCE.

ENTITY_CLASS: PATIENT.
    ASSOCIATES: DATA PATIENT NUMBER,
                DATA SAFE_FACTOR_RANGE FILE
                FACTOR_HISTORY.

DATA: SAFE_FACTOR RANGE.
    INCLUDES: DATA LOW_PRESSURE, DATA HI_PRESSURE,
              DATA LOW_TEMPERATURE, DATA HI_TEMPERATURE,
              DATA LOW_SKIN_RESISTANCE,
              DATA HI_SKIN_RESISTANCE.
    TRACED FROM: SENTENCE_4.

FILE: FACTOR_HISTORY.
    CONTAINS: DATA MEASUREMENT_TIME, DATA HPULSE,
              DATA HTEMPERATURE, DATA HBLOOD_PRESSURE,
              DATA HSKIN_RESISTANCE.
    TRACE FROM: SENTENCE_3.

ALPHA: EXAMINE_FACTORS.
    INPUTS: DATA DEVICE_DATA, DATA SAFE_FACTOR_RANGE.
    OUTPUTS: RANGE.
    DESCRIPTION: "THIS PROCESSING STEP FIRST RETRIEVES THE SAFE
                  FACTOR DATA ASSOCIATED WITH THE DEVICE, COMPARES
                  THE DEVICE DATA TO THE SAFE FACTOR RANGES FOR
                  THE PATIENT BEING MONITORED, AND DETERMINES
                  WHETHER THE FACTORS ARE IN_BOUNDS (RANGE_SAFE)
                  OR OUT_OF_BOUNDS".

DATA: PATIENT_DEVICE_NUMBER.
    INCLUDED IN: DATA SAFE_FACTOR_RANGE.
```

ELEMENTE in RSL

- Data-Segment

In diesem Segment werden die logischen Beziehungen (z.B. hierarchisch) zwischen Daten und ihr Zusammenhang mit den verschiedenen Komponenten des Systems beschrieben.

Als spezielle Typen gibt es z.B. INPUT-INTERFACE/OUTPUT-INTERFACE, durch welche die Ein-/Ausgabe Schnittstellen eines Teilsystems beschrieben werden können.

Den Daten können Attribute zugeordnet werden, z.B. MAXIMUM-VALUE, MINIMUM-VALUE, UNITS, INITIAL-VALUE, LOCAL, GLOBAL. Neben Objekten vom Typ INTEGER, REAL, BOOLEAN gibt es noch den Aufzählungstyp (ENUMERATION):

```
DATA: weekday.
   TYPE:  ENUMERATION.
   RANGE: monday, tuesday wednesday,
          thursday, friday.
```

- Validation-Segment

Mit Hilfe von R-Nets, ALPHAs und DATA-Segmenten werden die funktionellen Anforderungen beschrieben. Das Validations-Segment dient zur Beschreibung der Leistungsanforderungen (Performance). Durch Validationspunkte werden bestimmte Pfade in einem R-Net identifiziert und Validationspfade genannt. Validationspfade werden zur Überprüfung der Leistungsanforderungen herangezogen. In den Validationspunkten werden die zur Überprüfung notwendigen Daten gesammelt, um später einer Analyse unterzogen zu werden. Die Beschreibung eines Validationspfades besteht aus der Angabe der Folge seiner Validationspunkte. Zeitanforderungen können z.B. einem Validationspfad durch die Attribute MAXIMUM-

TIME und MINIMUM-TIME und die entsprechende Einheitenangabe UNITS zugeordnet werden. Komplexere Leistungsanforderungen, die abhängig sind von mehreren Validationspfaden, können auch analytisch durch algorithmische Beschreibungen von Testbedingungen in den Validationspunkten formuliert werden.

- Extensions-Segment

RSL ist erweiterbar; der Anwender kann neue Objekte, Relationen und Attribute definieren.

4.3.3 Prüfmittel von SREP

Aus der Sicht des Anwenders besteht der Analysator REVS von SREP aus folgenden Komponenten:

- Verarbeitung von RSL und etwaiger Erweiterungen
  Syntaxprüfung, Erstellung eines abstrakten Modells in der DB
- Interaktive Erstellung von R-Nets
  Graphische Ein-/Ausgabe (Menütechnik)

- Analyse des Beschreibungsmodells in der DB und Erzeugung von Analysereports

  - Prüfung auf Widerspruchsfreiheit, Vollständigkeit; dies ist auf verschiedenen Hierarchieebenen möglich (siehe PSL-Reports).

  - Prüfung der R-Net-Strukturen: Gibt es nur einen Startknoten? Enthalten die R-Nets nur die erlaubten Verzweigungsstrukturen?

  - Prüfung des Datenflusses durch die R-Nets: Werden lokale Daten benutzt, bevor sie initialisiert werden? Werden Daten in mehr als einem parallelen Pfad initialisiert?

  - Prüfung der Spezifikationsebenen: Sind alle Subnets definiert? Werden Subnets rekursiv benutzt? Alle ALPHAs und Subnets müssen mindestens in einem R-Net verwendet worden sein.

- Erzeugung einer Simulation

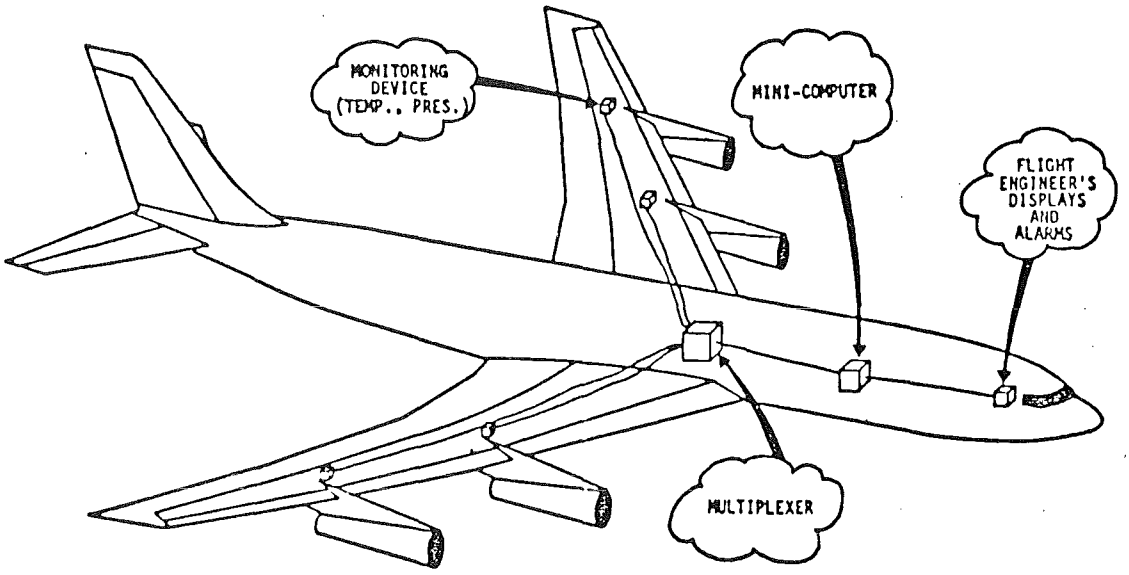
  Zur Überprüfung des dynamischen Systemverhaltens wird das in der DB befindliche abstrakte Modell der Systemanforderungen in ausführbaren Code für eine Simulation übersetzt. Dabei wird die Ablaufstruktur jedes R-Nets zur Entwicklung einer PASCAL-Prozedur benutzt, deren Kontrollfluß die R-Nets-Struktur implementiert. Jeder Prozeß (ALPHA) im R-Net wird ein Aufruf für eine Prozedur, die aus dem Modell (BETA) oder dem Algorithmus (GAMMA) für die durch ALPHA spezifizierten Anforderungen besteht. Die Datenstrukturen für die Simulation werden entsprechend ihrer Beschreibung in der DB synthetisiert.

- Ausführung der erzeugten Simulation

  Das oben erzeugte Programm kann mit Zeiten für den Start und das Ende der Simulation versehen werden. Entsprechend den in den Validationspunkten enthaltenen Testbedingungen können verschiedene Testanalysen durchgeführt werden (u.a. kann angegeben werden, welche Daten überwacht werden sollen und in welchem Validationspfad).

Enthalten die Testbedingungen z.B. Speicheranforderungen, so läßt sich die Planung von Overlay-Strukturen erleichtern. Datenflußschätzungen können die Analyse von Antwortzeiten unterstützen.

4.3.4 EXAMPLE - AN AIRCRAFT ENGINE MONITOR



(Nach Folien von B. Boehm, TRW)

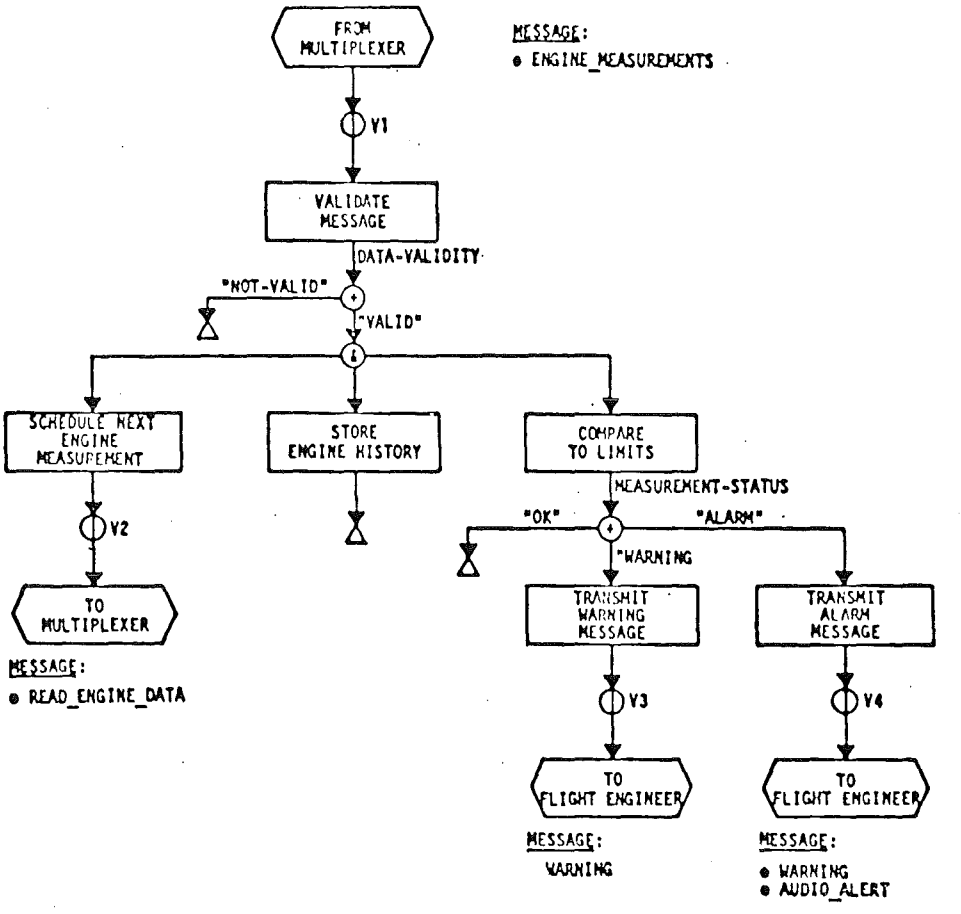
REQUIREMENTS SPECIFICATION

A MICROPROCESSOR AIRCRAFT ENGINE MONITOR FOR USE ON BOTH EXPERIMENTAL AND IN-SERVICE AIRCRAFT

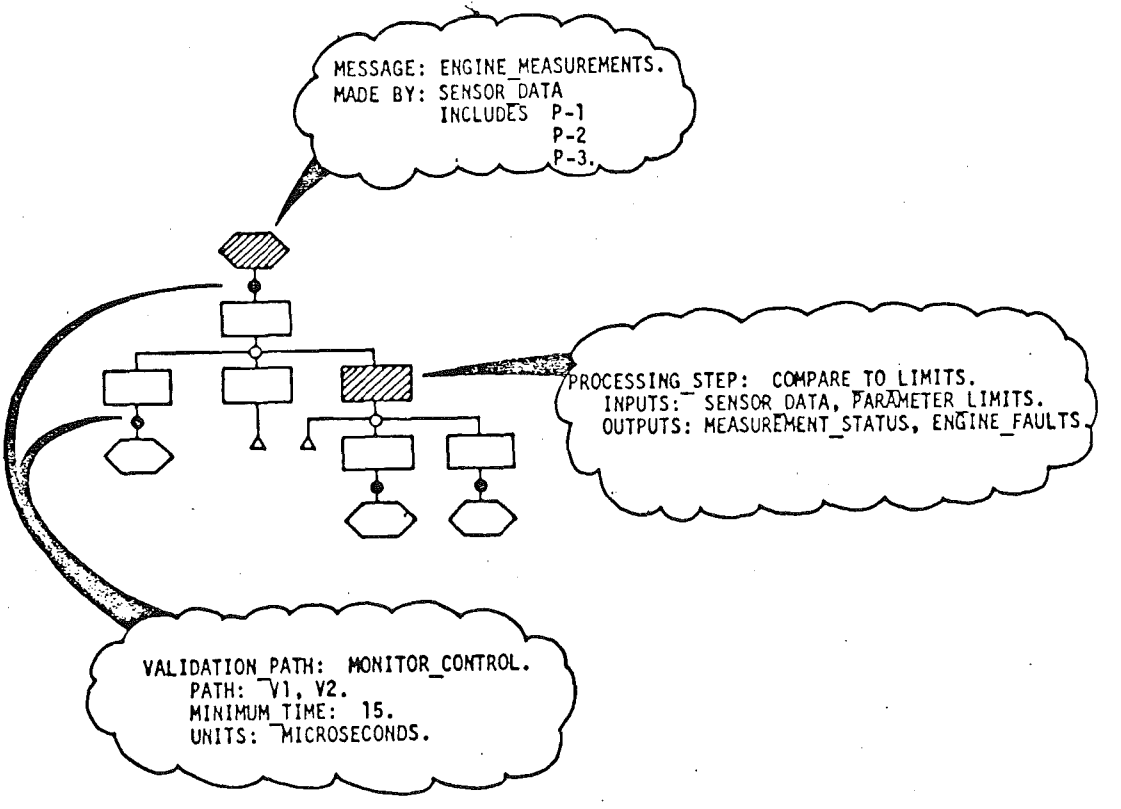
CAPABILITIES:

1. MONITOR 1 TO 10 ENGINES
2. MONITOR
   a. 3 TEMPERATURES (0 TO 1000° C)
   b. 3 PRESSURES (0 TO 4000 PSIA)
   c. 2 SWITCHES (OFF, ON)
3. MONITOR EACH ENGINE AT A SPECIFIED RATE (.1 TO 10 PER SEC)
4. OUTPUT WARNING MESSAGE IF ANY PARAMETER FALLS OUTSIDE PRESCRIBED LIMITS
5. OUTPUT WARNING MESSAGE AND ACTIVATE AUDIO ALARM IF ANY PARAMETER FALLS OUTSIDE PRESCRIBED LIMITS
6. RECORD HISTORY OF EACH ENGINE

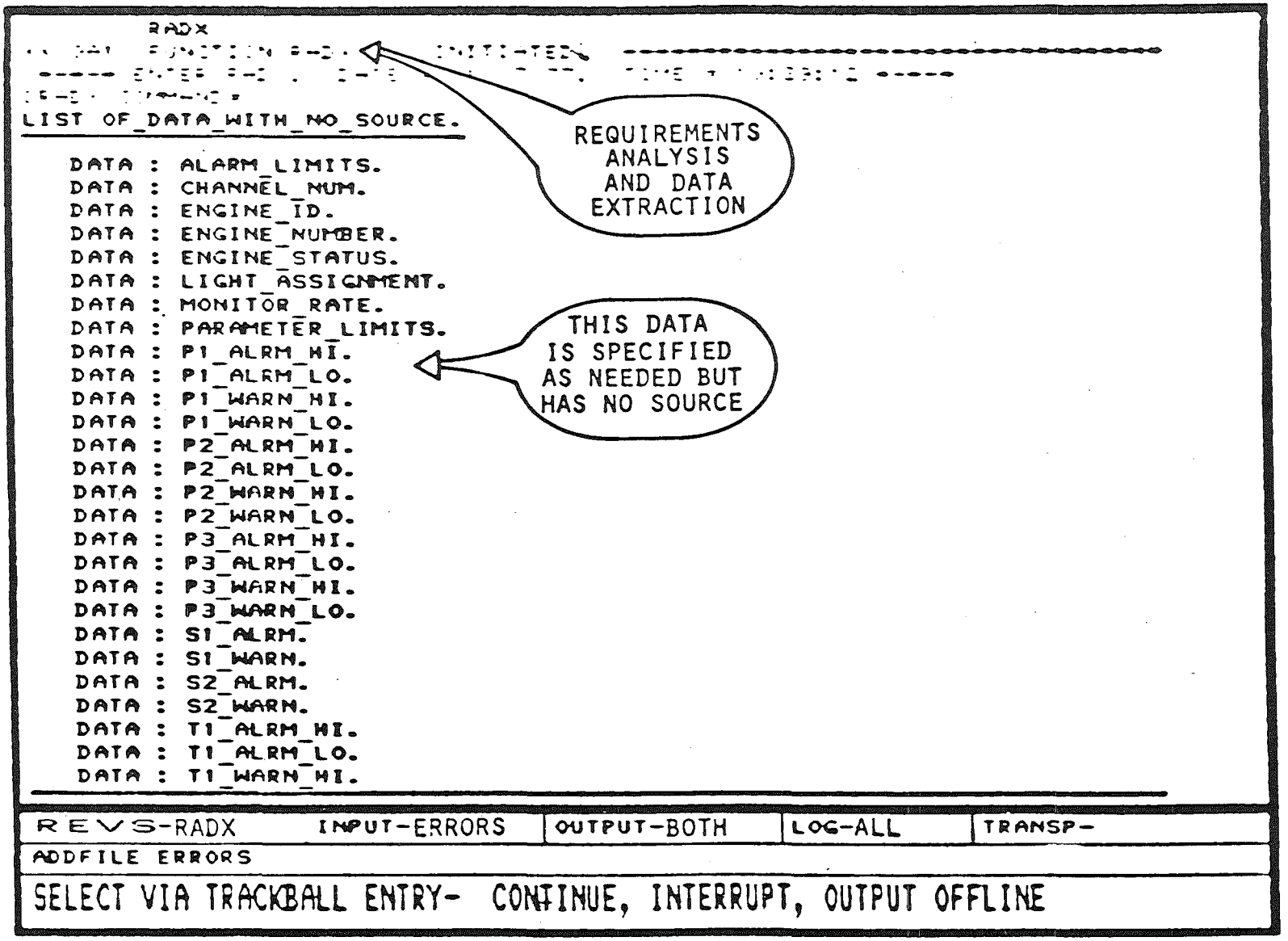
PROCESSING LOGIC IS DEFINED BY AN R-NET



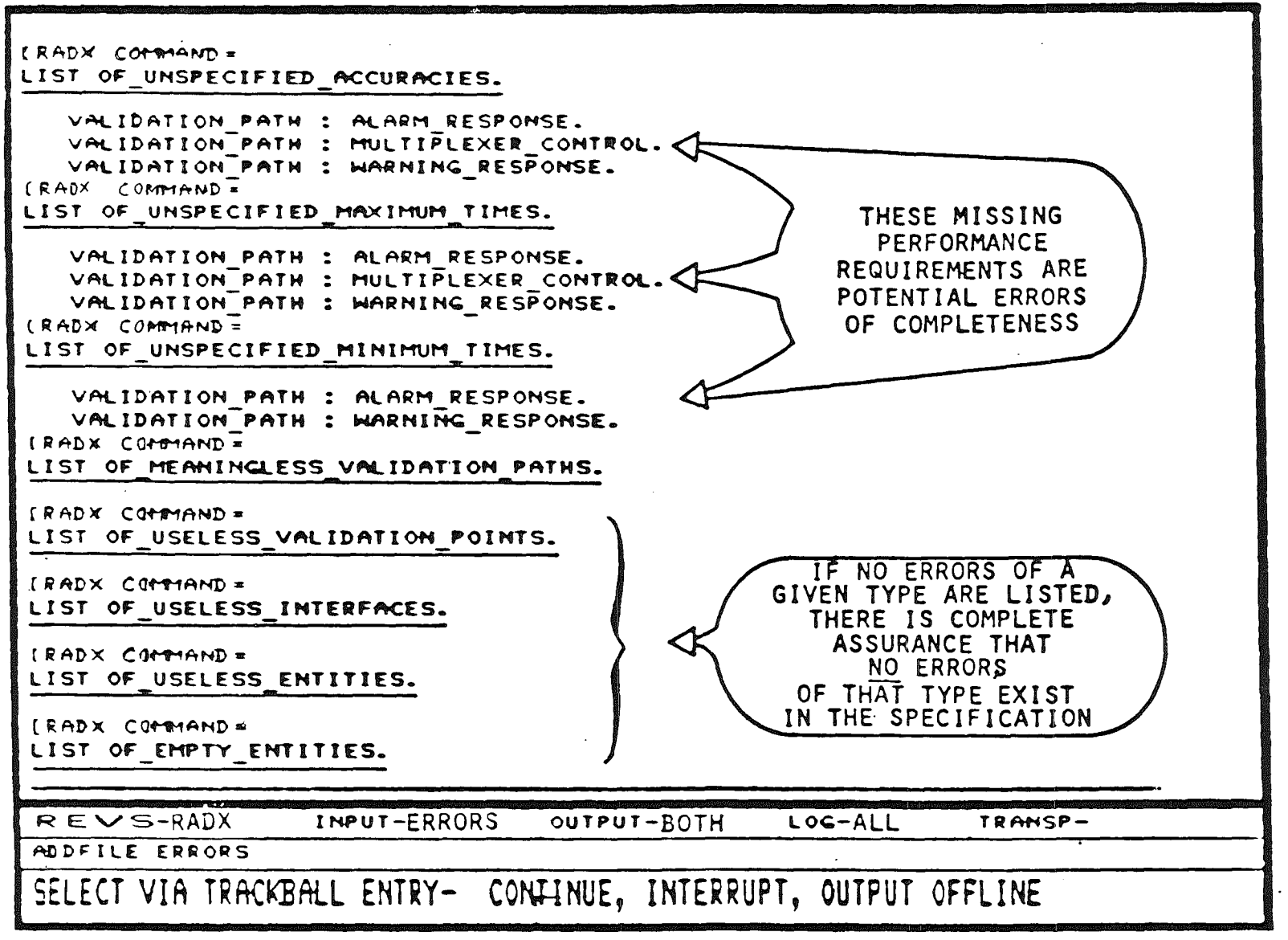
REQUIREMENTS DETAILS ARE SPECIFIED IN RSL



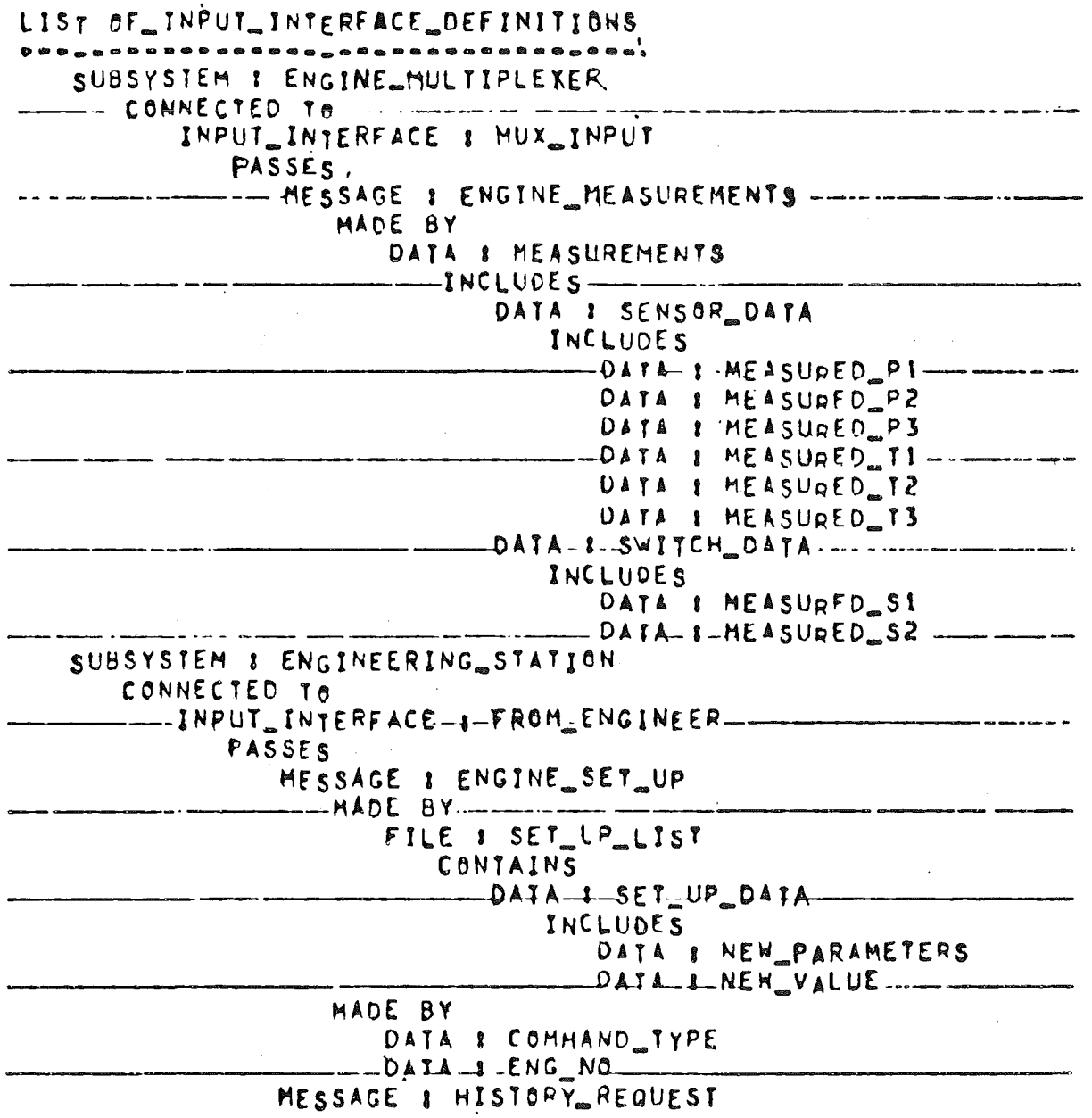
AUTOMATED ERROR CHECKS PROVIDE IMMEDIATE ANALYSIS OF THE REQUIREMENTS CONSISTENCY AND COMPLETENESS



AUTOMATED ERROR CHECKS (CONTINUED)



SAMPLE OF AUTOMATED DOCUMENTATION



AUTOMATED "CLASSICAL" SPECIFICATION FORMATS COULD BE PROVIDED

3.1.4.1 INPUT INTERFACE DEFINITION.

3.1.4.1.1 MUX INPUT. The Data Processor shall interface to the ENGINE MULTIPLEXER and shall be capable of receiving the ENGINE MEASUREMENTS messages.

3.1.4.1.1.1 ENGINE MEASUREMENTS MESSAGE. Each message of this type shall be made of SENSOR DATA and SWITCH DATA.

3.1.4.1.1.1.1 SENSOR DATA. This message component shall include the following information fields:

(a) MEASURED P1
(b) MEASURED P2
(c) MEASURED P3
(d) MEASURED T1
(e) MEASURED T2
(f) MEASURED T3.

3.1.4.1.1.1.2 SWITCH DATA. This message component shall include the following information fields:

(a) MEASURED S1
(b) MEASURED S2.

SOME BENEFITS OF USING SREM/SREP

- THE LANGUAGE AND METHODS PRECLUDE TYPICAL AMBIGUITIES, SUCH AS:
  - CAN THE ENGINES BE MONITORED AT DIFFERENT RATES?
  - DOES "OUTPUT WARNING" MEAN EACH TIME OR JUST THE FIRST TIME?
  - ARE "PRESCRIBED LIMITS" ESTABLISHED FOR EACH PARAMETER OR ARE THERE ONLY ONE MINIMUM AND ONE MAXIMUM?
  - ARE THE "PRESCRIBED LIMITS" FOR "WARNING" DIFFERENT THAN THOSE FOR "ALARM"?
- AUTOMATED ERROR CHECKS DETECT ERRORS OF CONSISTENCY AND COMPLETENESS, SUCH AS:
  - HOW ARE "PRESCRIBED LIMITS" DEFINED?
  - HOW QUICKLY MUST THE SYSTEM OUTPUT A "WARNING" OR "ALARM"?
  - WHAT DOES A "WARNING MESSAGE" CONTAIN?
  - WHAT IS TO BE DONE WITH THE HISTORY DATA FOR EACH ENGINE?
- AUTOMATED DOCUMENTATION PAYS HIGH DIVIDENDS
  - ONE COMPUTER RUN YIELDS COMPLETE SPECIFICATION WITH ALL CURRENT REVISIONS.
  - SAME THOROUGHNESS IS APPLIED TO EVERY REVISION -- LAST MINUTE PANICS DO NOT INTRODUCE UNDETECTED ERRORS.

5. Schnittstelle *) zu einem automatischen Testsystem

Auf dem Hintergrund der in den letzten Kapiteln diskutierten Beschreibungsmodelle für die Spezifikation und den Entwurf von Software sollen einige mögliche Verbindungen zu einem automatischen Testsystem skizziert werden.

Es lassen sich folgende Gesichtspunkte betrachten:

5.1 Unterstützung des Testsystems durch Informationen, die durch das Spezifikations-/Entwurfssystem in der DB bereitgestellt werden. Insbesondere soll die Auswahl von Testdaten aus der Programmstruktur erleichtert werden (Testdatenerzeugung).

5.2 Bereitstellung von Informationen, die als Vergleichsdaten zwischen dem Spezifikations-/Entwurfs-Modell und der Implementierung dienen
(Konsistenzprüfung der Implementierung).

5.3 Unterstützung des Entwurfs durch das Testsystem.

5.4 Sonstige Erweiterungen des Testsystems.

---

*) Die Ausführungen beziehen sich nur auf inhaltliche Aspekte der Schnittstelle, nicht auf deren Form. Es wird davon ausgegangen, daß sich die zur Verfügung gestellten Informationen in geeigneter Form in der DB befinden.

5.1 Testdatenerzeugung

5.1.1 Assertions (Zusicherungen)

- Ganz allgemein sind Assertions gültige Beziehungen zwischen Variablen. Sie können zur nicht-prozeduralen Beschreibung von Algorithmen dienen und somit bereits in der Spezifikations-/Entwurfsphase formuliert werden (siehe SSES). Die bei automatischen Programmbeweis-Systemen angewandten Methoden der Formulierung von Beziehungen zwischen Variablen (Assertions) und der Herleitung von Verifikationsbedingungen können in gewisser Weise zur Lösung des Problems der Testdatenerzeugung übertragen werden. Ausgehend von einer Zusicherung für die Eingangsvariablen zeigt man, daß die Folgerungen aus den Eingangsbedingungen zusammen mit den Anweisungen entlang den möglichen Wegen durch das Programm schließlich die Zusicherungen für die Ausgangsvariablen erfüllen. Die Verifikationsbedingungen für die einzelnen Pfade werden aus der Eingangszusicherung und den Anweisungen automatisch erstellt.

- Die Testdatenermittlung aus der Gemeinsamkeit von ähnlich durchlaufenen Teilpfaden wird durch Assertions erleichtert, da aus den Zusicherungen Testalgorithmen abgeleitet und den entsprechenden Stellen zugeordnet werden können (Retten von Zwischenzuständen).

- Meistens werden Assertions vorgeschlagen, die sich auf die Formulierung von logischen Ausdrücken zwischen Variablen beschränken (Stucky).

  Robinson und Roubine haben bei der Definition ihrer Entwurfssprache SPECIAL /RoRo 76/ (Specification and Assertion Language) um-

fangreichen Gebrauch von Assertions gemacht, z.B. bei der Formulierung von Zustandstransformationen, Fehlerbedingungen, invarianten Moduleigenschaften und Bedingungen, die zu einem bestimmten Zeitpunkt der Programmausführung erfüllt sein müssen. Die Sprache basiert auf dem Zustandsmodell von Parnas.

Eine Entwurfsbeschreibung in SPECIAL eröffnet weitere Möglichkeiten der Testunterstützung. (Der Anhang enthält eine Entwurfsbeschreibung in SPECIAL).

Folgende Fragen sind zu untersuchen:

- Können Assertions zur Bildung von Eingabeklassen herangezogen werden, die zu verschiedenen Aktionsfolgen führen?

- Wie können die zur Simulation verwendeten Validierungspfade und Validierungspunkte (SREP) des Entwurfsmodells zur Testunterstützung herangezogen werden, z.B. zur Erkennung von unabhängigen Programmteilen, Hilfestellung beim Abschätzen von "relevanten" Pfaden und beim Test von Zeitbedingungen?

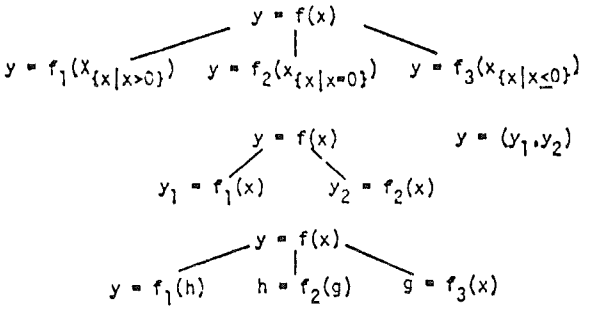
5.1.2 Entwurfsmethoden

Folgende Fragen und Ideen haben wir zu diesem Aspekt:

- Welchen Einfluß haben spezielle Entwurfsmethoden auf die Testbarkeit von Programmen (z.B. hierarchische Modularisierung => level by level testing)? Die Testbarkeit könnte Teil eines Qualitätsmaßstabes für Entwurfsmethoden sein.

- In letzter Zeit wurden Entwurfsmethoden entwickelt, die auf einer formalen Zerlegungsmethode beruhen. Z.B. beruht die Ent-

wurfsmethode HOS /HaZe 76/ auf sechs Axiomen, deren Verwendung eine notwendige und hinreichende Bedingung für einen Entwurf sind, der auf zuverlässigen Schnittstellen basiert. Aus diesen Axiomen lassen sich 3 gültige Zerlegungsstrukturen ableiten:



Man erkennt, daß bei diesen Zerlegungsstrukturen die Klassifizierung des Eingabebereichs einer Funktion (Modul) als Kriterium für die Zerlegung formal beschrieben wird. Da dies für den gesamten Entwurf gilt, können diese Informationen als Hilfe bei der Zerlegung des Eingabebereichs für das gesamte Programm in eine endliche Zahl von Äquivalenzklassen herangezogen werden und damit den Test unterstützen.

- Unter dem Begriff "structured design" /StMyCo 74/ werden verschiedene Bindungstypen für Moduln (functional, sequential, logical, coincidental etc.) eingeführt und Aussagen über die Spanne von Systemhierarchien (steil/flach) unter dem Aspekt des Entwurfs gemacht. Diese Aussagen müßten bzgl. ihrer Auswirkungen auf die Testbarkeit untersucht werden.

5.2 Daten aus dem Entwurf für den Test

Wenn das Entwurfssystem die entsprechenden Informationen bereitstellt, kann das Testsystem folgende Prüfungen ausführen:

- Vergleich der Ergebnisse der statischen Programm-Analyse mit den entsprechenden Informationen in der DB (Modulschnittstellen, cross-reference-listen, Programmstruktur).

- Vergleich der Eingangs-/Ausgangs-Assertions mit Ergebnissen des Programmtests (Wertebereiche, Invariante Prädikate).

- Vergleich der aus der Programmstruktur abgeleiteten Pfadprädikate mit den korrespondierenden, in der Spezifikation enthaltenen Zusicherungen.

- Vergleich der durch Simulation ermittelten (worst-case) Zeitbedingungen eines Validierungspfades mit entsprechenden Zeitmessungen des Programms anhand der Validierungspunkte (SREP).

- Spezifikation von Ausführungsfolgen und Ausführungsmöglichkeiten durch Assertions, verbunden mit path-expressions, und Überprüfung des aktuellen Programmverhaltens anhand der Spezifikation.
  Z.B. ASSERT LOCAL (DEF-A; REF-A *) *

  Eine Ausführungsfolge, die A enthält, muß mit der Definition von A beginnen, gefolgt von 0 oder mehr Ausführungen von A. Dieses Muster kann beliebig oft wiederholt werden /Ch 76/.

- In SREP werden die Anforderungen an ein System durch die Beschreibung der Eingabestimuli der zu durchlaufenden Pfade und der Angabe der Systemantworten spezifiziert. Nach Vorgabe entsprechender Eingaben können die daraus durch Test ermittelten Systemantworten mit den spez. Antworten verglichen werden und so zur Konsistenzprüfung zwischen Implementierung und Spezifikation beitragen.

5.3 Daten aus dem Test für den Entwurf

Folgende Informationen können vom Testsystem in die DB eingetragen werden und den Entwurf unterstützen:

- Aufwandsaussagen über gewünschte Systemänderungen
- Hinweise auf Systemoptimierungen aus der Ausführungshäufigkeit von Programmteilen (Dyn. Analyse).

- Entwurfsmethoden/Programmrichtlinien, um das Testen zu erleichtern, können aus den Testerfahrungen entwickelt werden.

- Ausgehend von einem funktionellen Simulationsmodell für den Entwurf (SREP), mit dessen Hilfe nur grobe Leistungsaussagen möglich sind, erhält man schrittweise genauere Aussagen, indem man immer mehr funktionelle "stubs" durch algorithmische Modelle ersetzt; vorausgesetzt, das Testsystem ist in der Lage, Leistungsmessungen in einer derartigen Umgebung (stubs, Code) durch einen geeigneten Testrahmen durchzuführen. Rückgemeldete Zeitmessungen könnten z.B. die Zerlegungsstruktur beeinflussen.

5.4 Sonstige Erweiterungen des Testsystems

Einige weitergesteckte Ziele für die Entwicklung von Testsystemen sind:

- Sprachunabhängigkeit (Syntaxgesteuert /No 75/).

- Test von realzeitspezifischen Fehlerarten (Zeitbedingungen, Prioritäten).

- Speicherung der getesteten Moduln zusammen mit Testdaten und Testresultaten in einer DB. Bei Änderungen von Moduln können "Abweichungsreports" erzeugt werden, aus denen man entnehmen kann, ob nur die beabsichtigten Änderungen gemacht wurden. Dadurch wird verhindert, daß bei der Korrektur von Fehlern neue Fehler entstehen.

- Test von Systemen, die "Stubs" enthalten, unvollständige Systeme (statisch/dynamisch).

- Test auf verschiedenen Sprachebenen (basic block, statement, expression, module, whole system).

- Verifikationsmethode, die auf den komplementären Eigenschaften des Programmtests und Programmbeweises beruhen (z.B. Verifikation von Synchronisationsschemata, Test von komplexen Programmstrukturen, die u.a. diese Schemata enthalten).

- Heuristische Testverfahren zur Erzeugung von Eingabedaten für den Test eines vorgegebenen Programmstücks.

- Teststrategien und ihre Relevanz zur quantitativen Bestimmung eines Maßes für Software-Zuverlässigkeit.

- Symbolischer Test.

- Trace der Testgeschichte auf Sprachebene, Testarchivierung.

- Testen der Robustheit von Programmen /Co 76/.

- Test von parallelen Programmen

  - auf Codeebene /RaKi 75/ (schwierig, da bei gleichen Eingabedaten die Fehler nicht reproduzierbar sind. Sie sollten deshalb durch konstruktive Maßnahmen verhindert werden),

  - auf Modellebene /CoQu 77/.

- Model-referenced-testing /Je 76/.

- Unterstützung bei der Interpretation von Testergebnissen (Welches Programmverhalten wird erwartet).

- Abhängigkeit der Testergebnisse vom Programmierstil.

6. Literatur

Abkürzung: SE 2 = Proc. 2nd Int. Conference Software Engineering San Francisco, CA, Oct. 13-15, 1976
(ein Teil der Papiere ist auch in den IEEE Transactions on Software Engineering, Vol. SE-3, no. 1, 2, 1977 erschienen)

/Al 76/	Alford, M.	A Requirements Engineering Methodology for real-time processing requirements, SE 2
/Au 76/	Austin, S.L. et.al.	SSL - A Software Specification Language, Science Applications Inc., Huntsville Report SAI-77-537-HU, 1976
/BeBi 76/	Bell, T.E. Bixler, D.C.	An extandable Approach to computer-aided software requirements engineering, SE 2
/Boe 77/	Boehm, B.W.	Software requirements and design aids Conference Reliable Software, INFOTECH, London, 1977
/Ch 76/	Chow, T.S.	A generalized assertion language, SE 2
/Co 76/	Cooper, D.W.	Adaptive Testing, SE 2
/CoQu 77/	Cooper, M.J. Quirk, W.J.	The Formal Specification of the requirements of complex real-time systems, (1977) unveröffentlicht
/DaVi 76/	Davis, C.G. Vick, C.R.	The Software Development System, SE 2
/HaZe 76/	Hamilton, M. Zeldin, S.	Higher Order Software - A Methodology for Defining Software, IEEE Trans. Software Engineering, Vol. SE 2, no. 1, 1976
/HoRy 76/	Hodges, B.C. Ryan, J.P.	A System for automatic Software evaluation, SE 2

/IrBr 76/ Irvine, C.A. Brackett, J.W. Automated Software Engineering through structured data mangement, SE 2

/Je 76/ Jessop, W.H. et.al. ATLAS - An Automated Software Testing System, COMPCON 1976 , S. 629-635

/No 75/ Noonan, R. Structured Programming and Formal Specification, Proc. 1st Nat. Conf. Software Engineering, Washington, Sept. 11-12, 1975

/RoRo 76/ Robinson, L. Roubine, O. SPECIAL - a SPECIfication and Assertion Language, Tech. Report Standford Research Institute, Sept. 1976

/TeHe 76/ Teichroew, D. Hershey, E.A. PSL/PSA: A computer-aided technique for structured documentation and Analysis of Information Processing Systems,SE 2

/RaKi 75/ Ramamoorthy, C.V. Kim, K.H. A Method of structuring and validating parallel programs, COMPCON 75, S. 219-222

/Sei 76/ Seifert, M. SADAT - Ein System zur automatischen Durchführung und Auswertung von Tests KFK-Ext. 13/75-5, Juni 1976

/StMyCo 74/ Stevens, W.P. Myers, G.J. Constantine, L.L. Structured Design IBM Systems Journal, Vol. 13, 2, 1974

Anhang:

Beispiel einer Entwurfsspezifikation in SPECIAL /RoRo 76/

siehe dazu Abschnitt 5.1.1 und Erläuterungen am Ende.

```
MODULE telephone_system

    TYPES

phone_id: DESIGNATOR;
phone_state:
{ hung_up, hung_up_but_connected, dial_tone, dialing,
 dialed_unconnected_number, ringing_another_phone, being_rung, connected,
 busy } ;
digit: { INTEGER i | 0 <= i AND i <= 9 };
phone_number_id: VECTOR_OF digit;

    DECLARATIONS

phone_id phone, phone1;
digit d;
phone_number_id phone_number;
phone_state ps;

    FUNCTIONS

VFUN state(phone) -> ps; $( state of "phone")
    HIDDEN;
    INITIALLY ps = ?;

VFUN connection(phone) -> phone1;  $( "phone1" is the phone that "phone"
                                      has dialed and is either connected or
                                      ringing)
    HIDDEN;
    INITIALLY phone1 = ?;

VFUN buffer(phone) -> phone_number;  $( sequence of digits dialed
```

```
                                        by "phone")
    HIDDEN:
    INITIALLY phone_number = ?;

VFUN valid_phone_number(phone_number) -> BOOLEAN b: $(TRUE for all valid
                                                      phone numbers)
    HIDDEN:
    INITIALLY TRUE $(initialized by the phone company):

VFUN directory(phone_number) -> phone:  $( "phone_number" that
                                          corresponds to "phone")
    HIDDEN:
    INITIALLY TRUE $( initialized by the phone company) :

OVFUN install(phone_number) -> phone:  $( creates a new designator
                                         "phone" that corresponds to
                                         "phone_number")
    EXCEPTIONS
       NOT valid_phone_number(phone_number);
       directory(phone_number) ~= ?;
    EFFECTS
       phone = NEW(phone_id);
       'directory(phone_number) = phone;
       'state(phone) = hung_up;
       'buffer(phone) = VECTOR();

OFUN disconnect(phone_number);  $( disconnects phone corresponding
                                  to "phone_number")
    DEFINITIONS
       phone_id phone IS directory(phone_number);
    EXCEPTIONS
       phone = ?;
       state(phone) ~= hung_up;
    EFFECTS
       'directory(phone_number) = ?;
       'state(phone) = ?;
       'buffer(phone) = ?;

OFUN pick_up_phone(phone): $( "phone" is picked up)
    EXCEPTIONS
       state(phone) = ?;
       NOT state(phone) INSET { hung_up, hung_up_but_connected, being_rung };
    EFFECTS
       IF state(phone) = hung_up
         THEN $( picking up to dial) 'state(phone) = dial_tone
       ELSE IF state(phone) = being_rung
         THEN $( answering phone)
            'state(SOME phone1 | connection(phone1) = phone) = connected
               AND 'state(phone) = connected
       ELSE $(resuming existing connection) 'state(phone) = connected;

OFUN dial(phone; d): $( dials a digit "d" from "phone")
    DEFINITIONS
       INTEGER j IS LENGTH(buffer(phone));
```

```
        phone_number_id newbuf
           IS VECTOR(FOR i FROM 1 TO j + 1
                     : IF i <= j THEN buffer(phone)[i] ELSE d);
        phone_id phone1 IS directory(newbuf);
    EXCEPTIONS
        state(phone) = ?;
    EFFECTS
        state(phone) INSET {dial_tone, dialing}
          => $( update buffer and change state) 'buffer(phone) = newbuf
          AND (IF phone1 ~= ?
            THEN $( a valid number has been reached) IF state(phone1) = hung_up
                THEN  $( ringing begins) 'state(phone) = ringing_another_phone
                      AND 'state(phone1) = being_rung
                     AND 'connection(phone) = phone1
                ELSE $( busy signal) 'state(phone) = busy
            ELSE IF valid_phone_number(newbuf)
              THEN $( not a connected number)
                   'state(phone) = dialed_unconnected_number
            ELSE 'state(phone) = dialing);

OFUN hang_up(phone); $( hangs up "phone")
    EXCEPTIONS
        state(phone) = ?;
        state(phone) INSET { hung_up, being_rung, hung_up_but_connected };
    EFFECTS
        IF EXISTS phone1 : connection(phone1) = phone
          THEN $(connection NOT terminated)
               'state(phone) = hung_up_but_connected
        ELSE $(back to original state)
            'state(phone) = hung_up
           AND 'buffer(phone) = VECTOR()
           AND (connection(phone) ~= ? => $(connected to someone else)
                'connection(phone) = ?
                AND 'state(connection(phone)) =
                 (IF state(phone) = ringing_another_phone
                    THEN $(ringing stops) hung_up
                  ELSE $(terminates connection) dial_tone));

END_MODULE
```

Erläuterungen zum Beispiel:

The example is a specification for a telephone system of a single area code. This is not a software system: it is implemented in hardware and its O-functions correspond to physical acts performed by people. However, the telephone system is a good example because everyone understands it, as opposed to most complex software systems. The system contains the following general design decisions:

- Telephones are named by a designator type (phone_id) rather than a telephone number. This corresponds to real life, in which a telephone is physically protected, i.e., knowing the number of a phone is not sufficient to be able to pick up that phone and dial from it.
- The mapping between phone numbers and connected phones is an invertible function. This is not true in a phone system with more than one area code, because a single phone number may identify different phones in different areas and because a phone is known by a different number when dialed from within the area than when dialed from outside the area.
- The state of a phone is indicated by the scalar type "phone_state". Based on the state of the phone, different things happen when an O-function is called. These states can later be mapped to particular switch positions in the actual phone circuits.
- The phenomenon that a connection can be terminated only by the party that initiated the call. Thus if the called party hangs up, the connection still exists, and the hung up phone has state "hung_up_but_connected".

There are also several features of SPECIAL whose use is worth pointing out:

- Use of a subtype to characterize a digit. Note that in this case there is no chance of misusing the subtype, since digits are only used with the equality operator.
- Use of vectors to represent phone numbers. The intensional vector constructor is used in the O-function "dial".
- Use of the SOME expression in the O-function "pick_up_phone". In this case there is only one value of "phone1" satisfying the given assertion.

The reader can see other ways of writing module specifications with the same properties as those above. The style to be chosen in writing specifications depends on one's desire for conciseness, readability, or even provability of the specifications. It is also possible to see that specifications may differ

considerably from their implementations. In fact, the difference between specification and implementation is often so great that specifications cannot be construed in any way as a guide to the programmer on how to write an efficient implementation. Such information must often be supplied separately from the module specification.