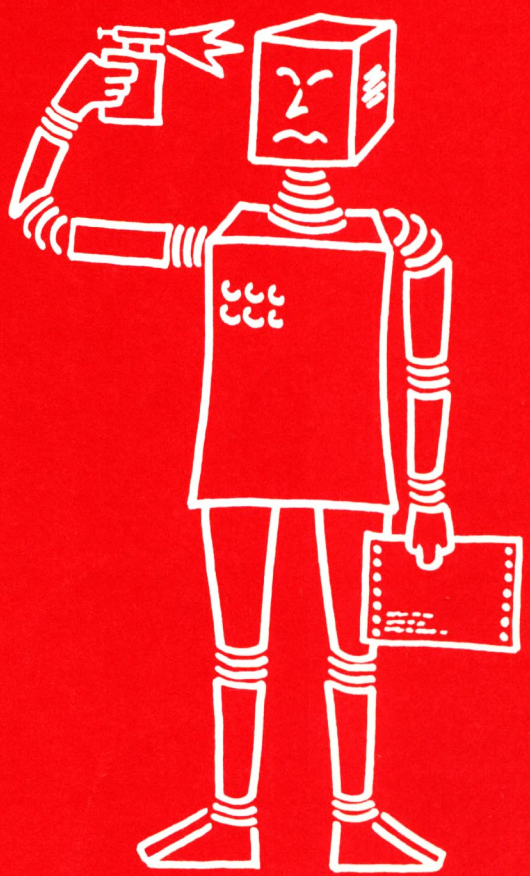
SEKI-PROJEKT

SEKI MEMO

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
West Germany

Universität Karlsruhe
Institut für Informatik I
Postfach 6380
D-7500 Karlsruhe
West Germany



SPESY -
Eine integrierte Softwareentwicklungs- und Verifikationsumgebung

V. Schölles, Ch. Beierle, A. Voß

MEMO SEKI-85-09 September 1985

SPESY -

Eine integrierte Softwareentwicklungs-

und Verifikationsumgebung

Memo SEKI 85-09
September 1985

V. Schölles, C. Beierle, A. Voß
Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
6750 Kaiserslautern
West Germany

Zusammenfassung

SPESY ist die Unterstützungsumgebung eines integrierten Softwareentwicklungs- und Verifikationssystems. Nach einer Einführung werden alle dem SPESY-Benutzer zur Verfügung stehende Kommandos beschrieben und das Verhalten des Systems an Hand von Dialogprotokollen erläutert.

Abstract

SPESY is the support environment of an integrated software development and verification system. Following an introduction, all commands being available for the user are described. The system's behaviour is illustrated using session protocols.

1. Einleitung

SPESY ist ein Entwicklungssystem für algebraische Spezifikationen (Specs) und ist die zentrale Komponente eines integrierten Programmentwicklungs- und Programmverifikationssystems ([BGGORV 83], [BV 85]). Diese Programmentwicklungsumgebung unterstützt es, Programme hierarchisch zu strukturieren, in kleinen Schritten zu entwickeln und zu verifizieren (verify-while-develop) , und sie stellt in allen Phasen des Softwareentwicklungszyklus , von der formalen Anforderungsspezifikation bis zum Pascalprogramm, geeignete Werkzeuge bereit. Dieser Entwicklungsprozeß läßt sich in eine Folge abstrakter und konkreter Schritte zerlegen (Abb.1).

Die abstrakten Schritte gehen von einer formalen Anforderungsspezifikation (axiomatisch) aus und enden in konstruktiv definierten Modellen (algorithmisch). Dabei kann man nochmals in Verfeinerungsschritte und Implementierungsschritte klassifizieren. Erstere ersetzen Teile von Spezifikationen durch präzisere Anforderungen und verkleinern somit die Menge der Modelle. So wäre die Ersetzung axiomatisch spezifizierter Operationen durch konstruktiv definierte Operationen ein Verfeinerungsschritt. Mit Implementierungsschritten werden abstrakte Spezifikationen durch konkretere ersetzt. Dabei wird die Menge der Modelle so gewechselt, daß alle Rechnungen in der alten Modellmenge durch Rechnungen in der neuen Modellmenge simuliert werden können.

Die konkreten Schritte setzen bei konstruktiv definierten Modellen an und reichen bis zum ausführbaren PASCAL-Programm. Auch hier kann man nochmals in Realisierungsschritte und Vorübersetzungsschritte aufteilen. Realisierungsschritte ersetzen konkrete Spezifikationen durch Module aus MODPASCAL, eine um das Modulkonzept erweiterte PASCAL-Version, sodaß sich die Spezifikationshierarchie in der Modulhierarchie widerspiegelt. Vorübersetzungsschritte erzeugen aus einem MODPASCAL-Modul ein PASCAL-Programm.

Alle abstrakten Schritte lassen sich in der Spezifikationssprache ASPIK ([BV 83a], [BV 85]) ausdrücken. Eine ASPIK-Spezifikation definiert durch ihre Signatur (Sorten- und Operationssymbole), ihre Formeln (Prädikatenlogik 1.Stufe) und ihre algorithmischen Definitionen eine Klasse von Algebren. Die Spezifikation kann hierarchisch strukturiert sein und kann instanziiert werden durch Ersetzung beliebiger, benutzter Spezifikationen [BGV 83]. Darüber hinaus ermöglicht ASPIK nicht nur die Definition von Spezifikationen, sondern auch von sogenannten Maps zur Beschreibung von Verfeinerungs- und Implementierungsbeziehungen. Aus diesem Grund ist ASPIK sogar als Spezifikationsentwicklungssprache anzusehen.

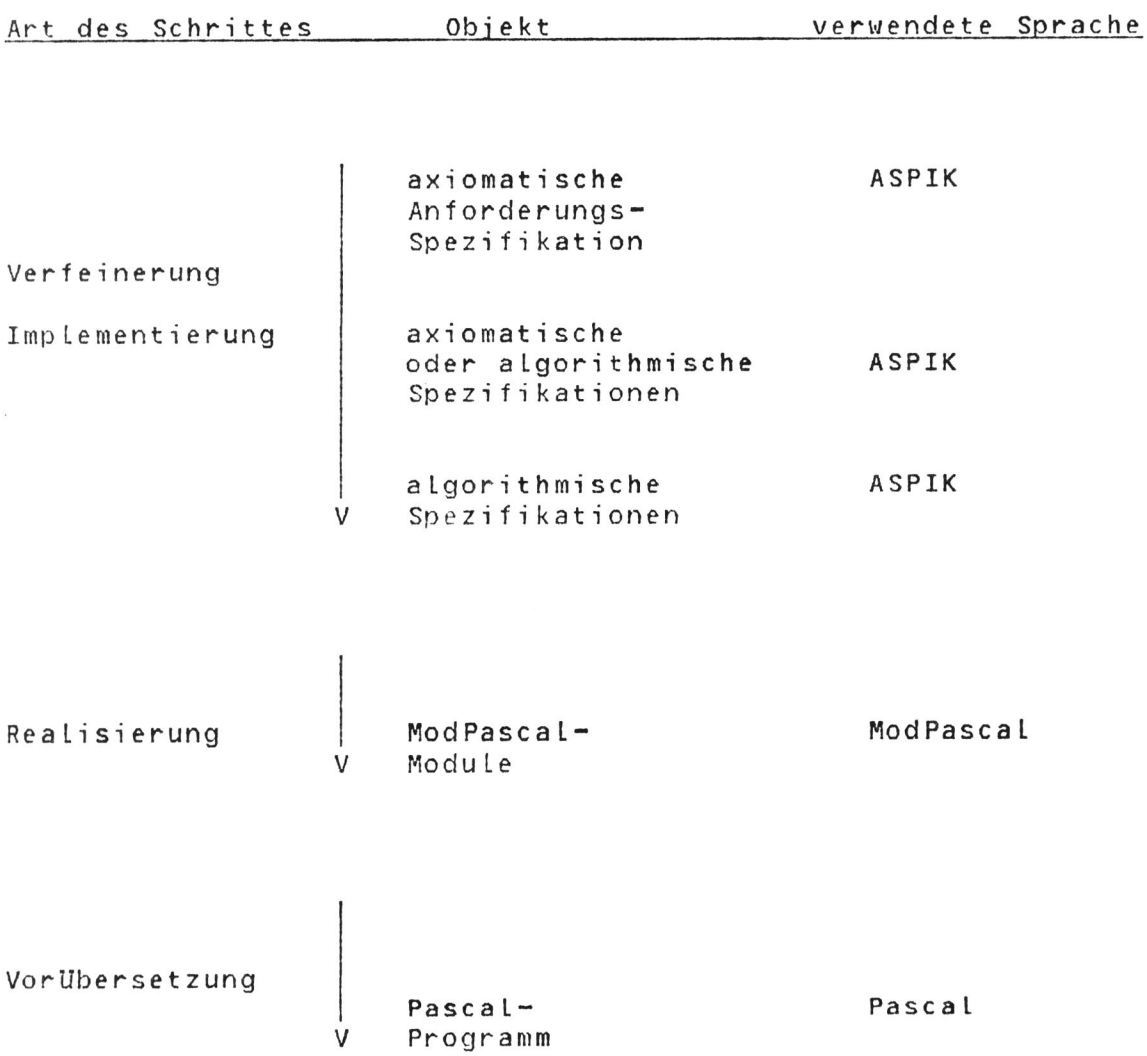


Abb.1 : Schrittweise Softwareentwicklung

Die beiden Strukturierungsmechanismen von ASPIK wurden bereits angedeutet:
Die Use-Beziehung für Spezifikationen und Maps ermöglicht den hierarchischen Aufbau dieser Objekte. Das bedeutet insbesondere für Spezifikationen, daß sie andere Spezifikationen und somit deren Sorten- und Operationssymbole benutzen können. Das Parameterisierungskonzept "parameterization-by-use" ([BV 83a], [BV 83b]) kennt keine explizite Parameterdefinition, sondern läßt alle benutzten Spezifikationen als Parameter zu. Die Parameterspezifikation wird erst zum Zeitpunkt der Instanziierung festgelegt.

Zur Speicherung dieser ASPIK-Objekte bietet SPESY ein Dateikonzept an, das zwischen Privatdateien, die der Benutzer anlegt, und Systemdateien, die eine für jeden Benutzer zugängliche Bibliothek darstellen, unterscheidet. Es gibt eine ausgezeichnete Systemdatei, SYS.BOOLFILE, die die ausgezeichnete Spezifikation BOOL enthält. Für jede andere Datei wird eine Umgebung (environment), bestehend aus Privat- und Systemdateien, definiert. Objekte von Systemdateien und solchen Privatdateien, die in der Umgebung anderer Privatdateien enthalten sind, können nicht manipuliert werden und sind korrekt. Innerhalb einer Datei sind alle darin definierten Objekte sichtbar und zusätzlich alle sichtbaren Objekte aus der Umgebung. Um die Konsistenz einer solchen Dateihierachie zu garantieren, müssen die darin existierenden Objektnamen voneinander verschieden sein.

SPESY ist ein Mehrbenutzersystem, das jedem Benutzer neben seinen Privatdateien auch die Systemdateien zur Verfügung stellt. Alle Dateien sind grundsätzlich gegen den Zugriff durch fremde Benutzer geschützt.

Neben interaktiven Systemen zum Eingeben und Editieren von ASPIK-Objekten stellt SPESY folgende Werkzeuge zur Verfügung:

- einen symbolischen Interpretierer zum Testen von konstruktiven Spezifikationen (zur Zeit noch nicht auf die neue Sprachversion umgestellt) [KRST 83].

- ein Programmtransformationssystem zum Auflösen von Rekursionen in Iterationen (zur Zeit noch nicht auf die neue Sprachversion umgestellt) ([Gei 84], [Pet 83], [Ge 83]).

- eine allgemeine Beweiserschnittstelle, über die Beweisaufgaben gestellt, ihre Lösung entgegengenommen und in SPESY verarbeitet werden können. Zur Zeit sind zwei Beweissysteme über diese Schnittstelle ansprechbar :
  Der Karlsruher Resolutionsbeweiser "Markgraf Karl Refutation Procedure" [BES 81] und ein in Bonn entwickeltes Termersetzungssystem [Tho 84], um Konsistenzbeweise durchzuführen.

- ein Subsystem zur Realisierung von konstruktiven ASPIK-Spezifikationen durch ModPASCAL bzw. PASCAL-Programme [BEORSW 85].

All diese Werkzeuge werden in SPESY auf verschiedenen Kommandoebenen (Level) angeboten, in Abhängigkeit von den manipulierten Objekten.

Das Spezifikationsentwicklungssystem SPESY ([BV 81/84], [BV 85]) ist in INTERLISP auf einer SIEMENS-Rechenanlage implementiert. Die vorliegende Implementierung entstand aus einer Vorgängerversion ([KRST 83], [Som 84]) durch umfangreiche Modifikationen und Ergänzungen um zum Teil völlig neue Komponenten, die von verschiedenen Leuten erstellt wurden ([Schö 85], [SPESY 85]).

2. Funktionen von SPESY

2.1. Aufgaben des Systemverwalters

Neben den registrierten Systembenutzern gibt es einen privilegierten Benutzer, dem als System-Manager die Systemverwaltung obliegt. Er ist der Eigentümer aller Systemdateien. Alle anderen Benutzer besitzen READ-Rechte für diese Dateien. Damit können sie Systemdateien in die Umgebung von Privatdateien aufnehmen und über den darin enthaltenen Spezifikationen und Maps neue Objekte aufbauen. Das READ-Recht erlaubt ihnen daneben auch das Kopieren von Systemdateien in Privatdateien.
Die Aufgaben des Systemverwalters sind :

1. Verwaltung der Systemdateien
2. Verwaltung der Benutzerliste, in der alle Personen, die das SPESY-System benutzen dürfen, registriert sind
3. Verwaltung der entsprechenden Passwortliste.

Der System-Manager kann Privatdateien der Benutzer in Systemdateien konvertieren, sofern die notwendigen Korrektheits- und Konsistenzbedingen erfüllt sind. Dem früheren Benutzer wird dabei das Eigentumsrecht über die Datei entzogen, und er erhält, wie alle anderen Benutzer auch, das READ-Recht auf die Datei.
Der System-Manager kann außerdem Systemdateien löschen, soweit sie nicht in der Umgebung einer anderen Datei enthalten sind.

Er kann neuen Benutzern das System verfügbar machen, indem er für diese einen neuen Eintrag in die Benutzer- und Passworttabelle erstellt. Für einen bereits bekannten Benutzer hat er auch die Möglichkeit, dessen Passwort zu verändern.

Der System-Manager kann Benutzern das Recht auf die Systembenutzung entziehen. Alle Dateien, sowie alle vorhandenen Rechte auf System- oder auf Privatdateien dieser Benutzer werden gelöscht.

2.2. Dem Benutzer zur Verfügung stehende Funktionen

2.2.1. Dateibezogene Funktionen

Jeder Benutzer des Systems hat die Möglichkeit, beliebig viele Privatdateien, deren Eigentümer er wird, neu anzulegen und diese auch wieder zu löschen. Privatdateien können durch explizites Eröffnen oder durch Kopieren erstellt werden. Dateien sind entweder leer oder enthalten ASPIK-Objekte, d.h. Spezifikationen oder Maps. Um Namenskonflikte zu vermeiden, trägt jede Datei die Benutzerkennung durch einen Punkt getrennt als Präfix. Der Benutzer darf dieses Präfix aber nur dann angeben, wenn sich ein Kommando auf eine fremde Datei bezieht, sonst nicht.

Dateien sind vor dem zerstörenden Zugriff durch fremde Benutzer geschützt. Der Benutzer kann explizit Zugriffsrechte für seine Dateien vergeben. Dabei kann er zwischen READ- und MESSAGE-Recht unterscheiden. Das READ-Recht erlaubt einem fremden Benutzer eine Datei zu kopieren. Das MESSAGE-Recht gestattet zusätzlich mit Hilfe eines Message-Systems Nachrichten bzgl. der Datei zu hinterlegen. Um die Objekte einer fremden Privatdatei benutzen oder editieren zu können, muß diese zuerst kopiert werden. Dieses Konzept stellt sicher, daß einerseits andere Benutzer die Privatdateien eines Benutzers nicht zerstörten können, andererseits aber alle Objekte dieses Benutzers von den anderen genutzt werden können.

Der Benutzer kann die Umgebung seiner Dateien verändern. Dies ist allerdings nur unter bestimmten Voraussetzungen erlaubt, um die Konsistenz der Dateihierachien nicht zu verletzen. Er kann Privatdateien oder Systemdateien in eine bestehende Umgebung neu aufnehmen, oder auch solche Dateien aus der Umgebung entfernen. Daneben gibt es die Möglichkeit, einzelne Objekte innerhalb einer Dateihierachie von oben nach unten zu verschieben. Eine letzte Variante zum Manipulieren einer Dateiumgebung ist das Verschmelzen aller Dateien der Umgebung. Damit kann der Benutzer eine gesamte Umgebung in einer Datei zusammenfassen, wobei alle Beziehungen zwischen den Objekten erhalten bleiben.

Das System bietet dem Benutzer vielfältige Informationsmöglichkeiten über Dateien an. Der Benutzer hat die Möglichkeit, sich die Namen seiner Dateien auflisten zu lassen oder sich Auskunft einzuholen, welche Objekte auf einer Datei vorhanden sind. Er kann sich eine Datei ausdrucken lassen oder sich über deren Umgebung informieren. Er kann sich über die Zugiffsrechte für Dateien unterrichten lassen, die er an andere Benutzer vergeben hat, bzw. die er von anderen Benutzer erhalten hat. Hinzu kommt, daß man Nachrichten für Dateien absetzen, lesen oder auch löschen kann.

2.2.2. Funktionen für ASPIK-Objekte

SPESY unterstützt die Eingabe von ASPIK-Objekten. Diese können anschließend editiert oder wieder gelöscht werden. Um die kontextsensitive Korrektheit von editierten Objekten feststellen zu können, steht dem Benutzer ein Checker zur Verfügung. Darüberhinaus kann der Benutzer auch semantische Eigenschaften untersuchen, indem er mit den angeschlossenen Beweisern arbeitet. Objekte, sowie deren Beziehungen und Schnittstellen untereinander, können auf den Bildschirm oder Drucker ausgegeben werden. Ein Objekt kann von einer Datei in eine andere kopiert werden. Ferner kann der Benutzer nachsehen, ob ein Objektname in einer seiner Dateien bzw. in einer zugreifbaren Datei eines fremden Benutzers auftritt. Konstruktive Teile einer Spezifikation können interpretiert und darin auftretende Rekursionen in Iterationen transformiert werden. Der Anschluß einer Realisierungskomponente an SPESY erlaubt es, für Objekte Realisierungen und schließlich PASCAL-Programme zu entwerfen.

2.2.3. Allgemeine Funktionen

Das System bietet dem Benutzer die Möglichkeit, eine Dialogsitzung ganz oder teilweise protokollieren zu lassen. Zu Beginn einer Sitzung ist das Protokoll immmer eingeschaltet, und am Ende wird der Benutzer vom System gefragt, ob es ausgedruckt werden soll. Die durch das Protokoll erzeugte Ausgabedatei ist nicht durch SPESY-Kommandos manipulierbar.

Ein HELP-Kommando informiert den Benutzer über die verfügbaren Kommandos. Er kann sich sowohl eine Tabelle dieser Kommandos, als auch eine kurze Beschreibung eines einzelnen Kommandos auf den Bildschirm ausgeben lassen.

3. Struktur von SPESY aus Benutzersicht

Die unter 1. und 2. genannten Konzepte und Funktionen bestimmen die Struktur von SPESY und spiegeln sich in den ausführbaren Kommandos wider. Die Realisierung der Konzepte und Funktionen ist in SPESY auf verschiedene Ebenen (Level) verteilt. Die Ebenen sind hierachisch angeordnet. Die Kommandos zum Betreten der verschiedenen Ebenen und die Hauptaufgabe einer jeden werden im folgenden kurz beschrieben. Zum Verlassen der Ebenen gibt es grundsätzlich das END-Kommando. Daneben kann man einige Level auch durch das ABORT-Kommando abbrechen.

Die oberste Ebene ist der Systemlevel. Dort ist einerseits die Systemverwaltung, andererseits das Dateimanagement realisiert. Die Kommandos zur Verwaltung des Systems stehen nur dem System-Manager zur Verfügung. Ansonsten sind alle Kommandos, die der Verwaltung der Benutzerdateien und deren Zugriffsrechte dienen, allgemein verfügbar. Die Objekte, die der Systemlevel verwaltet, sind Dateien bzw. Dateihierachien.

Die Ebene unterhalb des Systemlevels ist der Filelevel. Dorthin gelangt man durch das OPEN-Kommando des Systemlevels. Der Benutzer kann zwischen zwei Varianten wählen. Entweder betritt er den Filelevel mit einer bereits existierenden oder mit einer neuen Datei. Im ersten Fall sind auf dem Filelevel alle Objekte der geöffneten Datei und deren Umgebung sichtbar, wobei die Objekte der Umgebung allerdings nicht manipulierbar sind. Im anderen Fall muß der Benutzer eine Umgebung für die neue Datei definieren (mindestens SYS.BOOLFILE). Die Objekte, die der Filelevel verwaltet, sind die Objekte der geöffneten Datei, d.h. Spezifikationen (Specs) oder Maps.

Unterhalb des Filelevels gibt es mehrere parallele Ebenen. Es gibt jeweils einen Editor für Specs und für Maps. Unterhalb der beiden Editorlevel gibt es jeweils einen Inputlevel (Eingabeebene). Die Editoren betritt man vom Filelevel aus durch das EDIT-Kommando, wobei man im Kommando spezifiziert, welchen Editor man betreten will und welches Objekt editiert werden soll. Im Editor können die einzelnen Klauseln dieses Objekts verändert werden, ohne daß kontextsensitive Überprüfungen vorgenommen werden. Um diese Überprüfungen durchzuführen, steht ein Checker zur Verfügung, der vom Editor (und vom Filelevel) aufgerufen werden kann.

Der Inputlevel für Specs bzw. Maps kann entweder direkt vom Filelevel betreten werden, wobei implizit der jeweilige Editorlevel betreten und dann in den Inputlevel verzweigt wird, oder vom darüber liegenden Editorlevel aus. Das Kommando dazu ist jeweils das INPUT-Kommando. Der Inputlevel für Specs kann sowohl im Dialog als auch von einer Datei Spezifikationen bzw. Teile davon einlesen. Die in beiden Inputlevel eingegebenen Objekte werden automatisch auf Korrektheit überprüft und nur akzeptiert, falls sie sowohl kontextfrei als auch kontextsensitiv korrekt sind. Die Beschreibung der Korrektheitsbedingungen für Specs ist in [Lic 85] zu finden, die für Maps in

[Spa 85]. Nach dem Verlassen des Inputlevels befindet man sich im Filelevel, falls man ein vollständiges Objekt eingegeben hat, bzw. im jeweiligen Editorlevel , falls das eingegebene Objekt noch nicht vollständig war. Will man Klauseln eines Objekts eingeben, ohne daß sofort kontextsensitive Korrektheitsüberprüfungen durchgeführt werden, so gibt man diese Klauseln auf dem jeweiligen Editorlevel ein.

Der Benutzer kann die beiden Editorlevel auf zwei Arten verlassen. Einmal mit dem END-Kommando, wodurch alle bis dahin gemachten Änderungen am Objekt abgespeichert werden, zum anderen mit dem ABORT-Kommando, wodurch keine Änderung gemerkt wird.

Neben den oben genannten Level kann man vom Filelevel aus den Realisierungslevel betreten. Dieses Subsystem ist mit dem RL-Kommando erreichbar. Dort können aus konstruktiven Specs und mit Hilfe von Maps über mehrere Zwischenschritte bezüglich der Spezifikation korrekte PASCAL-Programme erstellt werden.

Vom Filelevel sind weitere Werkzeuge bzw. Schnittstellen verfügbar. Mit dem PROVE-Kommando kann der Benutzer die Schnittstellen zu dem Resolutionsbeweiser "Markgraf Karl Refutation Procedure" (MKRP) und dem "Rewrite Rule Labor" (RRLAB), einem Termersetzungssystem, herstellen. Darüber können Terminierungs-, Konsistenz- und Herleitbarkeitsbedingungen als semantische Korrektheitsbedingungen für ASPIK-Objekte nachgewiesen werden. Zur Zeit kann zu Testzwecken auch noch der Benutzer diese Eigenschaften festlegen und damit Ergebnisse manueller Beweise akzeptieren. Die Programmtransformationsebene und der Interpreterlevel sind für die Vorgängerversion voll ausgearbeitet und funktionsfähig; in der vorliegenden Version sind sie im Moment aber nicht erreichbar, da sie noch nicht auf die aktuelle ASPIK-Syntax umgestellt sind.

Die Struktur von SPESY und die Aufrufkommandos der Ebenen sind in den Abbildungen 2 und 3 dargestellt.

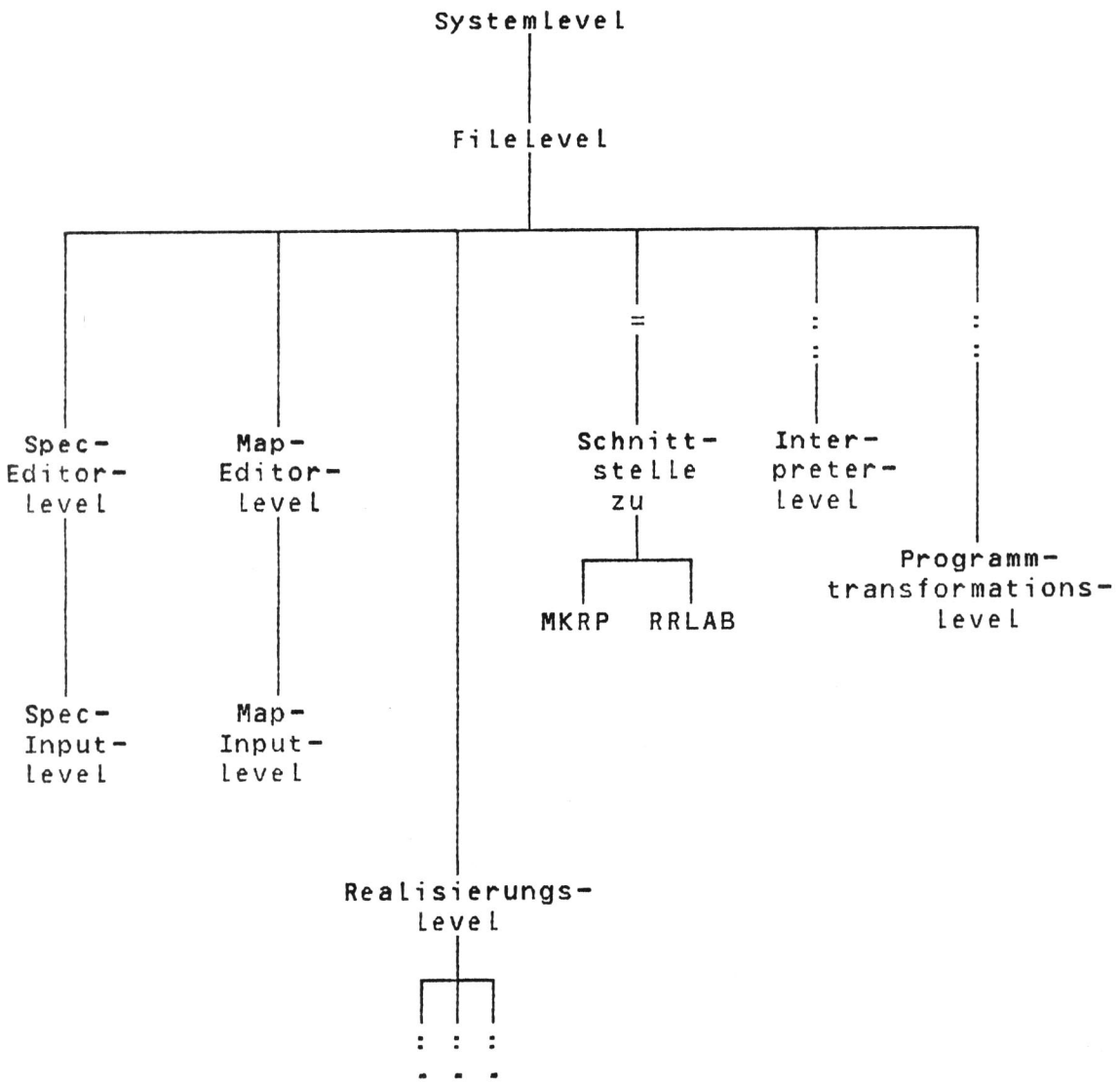
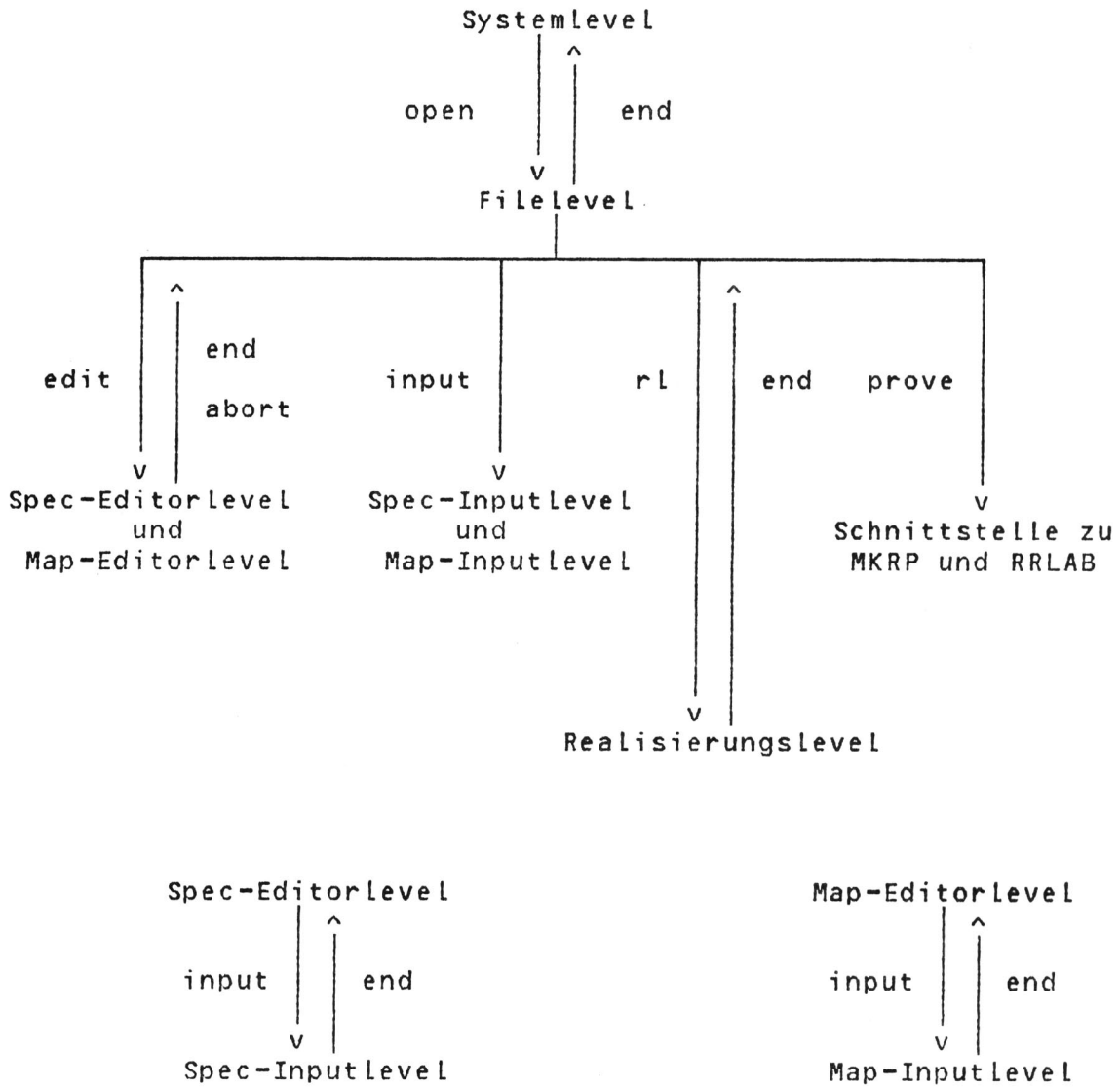


Abb.2: Kommandoebenen in SPESY



Hinweis:
1. Man gelangt vom Filelevel mit dem Edit-Kommando auf die beiden Inputlevel, falls das im Kommando angegebene Objekt neu ist.
2. Von den beiden Inputlevel gelangt man auf den Filelevel, falls ein Objekt korrekt und vollständig eingegeben wurde.
3. Falls man auf dem Filelevel das Prove-Kommando aufruft, wird man anschließend auf den Systemlevel zurückgesetzt.

Abb.3: Aufruf- und Beendigungskommandos der SPESY-Level

4. Benutzung des Systems

4.1. Systemaufruf

Der Aufruf von SPESY aus der BS2000-Betriebssystemebene erfolgt durch das Kommando **/DO SPESY.** Dorthin kommt man wieder durch das Systemlevelkommando **END** zurück. Der Aufruf bewirkt, daß SIEMENS-INTERLISP [Int] und SPESY geladen werden. Der Benutzer muß sich gegenüber dem System ausweisen, damit dieses feststellen kann, ob er als SPESY-Benutzer registriert ist. Das System fragt nach seiner **Benutzerkennung** und seinem **Passwort.** Ist die Eingabe korrekt, wird das Sitzungsprotokoll eingeschaltet und der Benutzer gelangt auf den Systemlevel. Sollte zur gleichen Zeit ein anderer Benutzer mit dem System arbeiten, erscheint folgende Warnung auf dem Bildschirm:

```
*****************************************
* SPESY IN USE - DON'T CHANGE ANYTHING *
*****************************************
```

In diesem Fall kann der Benutzer keine Kommandos absetzen, die eine Veränderung der Systemverwaltungslisten (SYSTEMLOG) zur Folge hätten. Auf dem Systemlevel können daher dann folgende Kommandos nicht ausgeführt werden:
CONVERT, COPY, COPYFILE, DELETE, DELETEUSER, ENVIRONMENT, MERGEFILE, MOVE, Anlegen neuer Dateien mit OPEN, RIGHT und **USER.**
Auf dem Filelevel werden die Kommandos **RL** und **PROVE** abgewiesen. Daneben gilt für **CHECK, DELETE, EDIT** und **INPUT,** daß sie zwar ausgeführt, die dabei gemachten Änderungen aber nicht abgespeichert werden. Eine entsprechene Warnung wird ausgegeben. Das Arbeiten mit SPESY ist in diesem Fall nur zu Testzwecken sinnvoll.

SPESY meldet sich mit:

SPECSYSTEM (VERS 2.0) GENERATED ON <date><time> IS READY:
PROTOCOL STARTET.
SYSTEMLEVEL:

und erwartet die Eingabe von Systemlevel-Kommandos.

4.2. Allgemeine Kommandokonzepte

Verlangt ein Kommando Parameter, so können diese direkt vom Benutzer angegeben werden. Fehlen die Parameterangaben, so werden sie vom System angefordert. Fast alle Kommandos besitzen festgelegte Abkürzungen. Jedes SPESY-Kommando kann durch ein Semikolon abgeschlossen werden. Die Kommandos, die mit einem Semikolon beendet werden müssen, haben eine variable Zahl von Parametern. Alle anderen Kommandos besitzen eine feste Parameterzahl. Da alle Kommandos eindeutig beendet werden müs-

sen, ist es möglich, mehrere Kommandos in einer Zeile einzugeben (Kommandofolge). Ist ein Kommando innerhalb einer Folge fehlerhaft, so werden alle darauffolgenden Kommandos ignoriert.

In SPESY existieren sogenannte Kontrollparameter, die explizit abgefragt und verändert werden können. Sie beeinflussen die Ausführung verschiedener Kommandos. Die Kontrollparameter und ihre Bedeutung sind in 4.3.2.2. in der Beschreibung des CONTROL-Kommandos festgehalten.

Daneben gibt es auch sogenannte Dialogparameter, die nicht explizit abgefragt oder verändert werden können. Auf sie kann der Benutzer in einem Kommando an geeigneten Stellen zugreifen, indem er den Parameter @ angibt. Die Dialogparameter werden als Wert von @ automatisch vom System eingesetzt. Es gibt folgende Dialogparameter:

- USER : Ist die Benutzerkennung, die nach dem Aufruf von SPESY eingegeben wurde.
- FILE : Ist die Datei, mit der zuletzt der Filelevel betreten wurde. FILE ist solange undefiniert, bis ein erfolgreiches OPEN-Kommando abgesetzt wird.
- OBJECT: Ist das Objekt, mit dem zuletzt ein Editor- oder Inputlevel betreten wurde. OBJECT ist solange undefiniert, bis zum ersten Mal ein EDIT- oder INPUT-Kommando abgesetzt wird.

Es gibt in bestimmmten Befehlen Parameter, die einen * als Alternative zulassen. Die Bedeutung von * entspricht dem Wort 'alle' . Kann der Benutzer zum Beispiel zwischen <userid> und * wählen, bedeutet die Angabe von * , daß alle Benutzer des Systems betrachtet werden.

4.3. Befehle des Systemlevels

Auf dem Systemlevel gibt es zwei Kommandoarten:

- priviligierte Kommandos, die nur vom Systemmanager ausgeführt werden können;
- nichtpriviligierte Kommandos, die alle Benutzer gebrauchen können.

4.3.1. Übersicht

Privilegierte Kommandos

CONVERT				
DELETEUSER	alias	**DELU**		
USER	alias	**U**		
USER?	alias	**U?**		

Nichtpriviligierte Kommandos

CONTROL	alias	**C**		
CONTROL?	alias	**C?**		
COPY				
COPYFILE	alias	**COPYF**		
DELETE	alias	**DEL**		
DELETEMESSAGE	alias	**DELM**		
DIRECTORY	alias	**DIR**		
END				
ENVIRONMENT	alias	**ENV**		
ENVIRONMENT?	alias	**ENV?**		
FIND	alias	**F**		
HELP	alias	**H**	alias	**?**
LISTFILES	alias	**LISTF**	alias	**LF**
MERGEFILE	alias	**MF**		
MESSAGE	alias	**M**		
MESSAGE?	alias	**M?**		
MOVE				
OPEN	alias	**O**		
PRINT				
RIGHT	alias	**R**		
RIGHT?	alias	**R?**		

4.3.2. Beschreibung der Kommandos

In der nachfolgenden Kommandobeschreibung werden nichtterminale Parameter in spitze Klammern eingeschlossen, z.B. bedeutet <userid> die Angabe einer Benutzerkennung. Terminale Parameter sind durch Großbuchstaben gekennzeichnet. Alternativen eines Parameters sind durch einen senkrechten Strich getrennt. USER bezeichnet im folgenden immer den aktuellen Benutzer des SPESY-Systems. Der Parameter <objectid> in einem der Kommandos **COPY FIND MOVE** bezeichnet immer entweder einen Specid oder ein Mapatom. Soll ein Parameter eine Datei eines fremden Benutzers bezeichnen, so ist immer der Dateiname mit vorangestellter Benutzerkennung anzugeben.

4.3.2.1. Beschreibung der priviligierten Kommandos

Kommandoname: **CONVERT**

Parameter : <fileid_1>
<fileid_2>

Dieses Kommando wandelt eine private Datei von USER in eine Systemdatei um. Die private Datei wird durch den ersten Parameter spezifiziert, der Name der neuen Systemdatei durch den zweiten. Drei Bedingungen müssen für eine erfolgreiche Umwandlung erfüllt sein.

a) Die Umgebung der privaten Datei darf nur aus Systemdateien bestehen.
b) Alle Objekte der privaten Datei müssen ok sein.
c) Alle Objektnamen der privaten Datei müssen sich von den Namen aller Systemobjekte unterscheiden.

Alle Benutzer erhalten READ-Recht für die neue Systemdatei. Die private Datei wird gelöscht, und in der Umgebung aller Dateien wird der alte Name durch den neuen ersetzt.

Kommandoname: **DELETEUSER** alias **DELU**

Parameter : <userid>

Dieses Kommando entzieht dem spezifizierten Benutzer die Erlaubnis, mit dem SPESY-System zu arbeiten. Alle Dateien dieses Benutzers werden gelöscht, und seine Rechte an fremden Dateien werden entzogen.

Kommandoname: **USER** alias **U**

Parameter : <userid> (max. 8 Zeichen)
<password> (max. 8 Zeichen)

Dieses Kommando erlaubt einem neuen Benutzer mit dem SPESY-System zu arbeiten. Der neue Benutzer erhält Leserecht für alle Systemdateien. Mit diesem Kommando kann auch das Passwort eines bereits bekannten Benutzers verändert werden. Dazu gibt man dessen Benutzerkennung und das neue Passwort an.

Kommandoname: **USER?** alias **U?**

Parameter : --

Dieses Kommando listet alle dem System bekannten Benutzer und deren Passwörter.

4.3.2.2. Beschreibung der nichtpriviligierten Kommandos

Kommandoname: **CONTROL** alias **C**

Parameter : <controlpar_1>
:
.
<controlpar_n> ;

Für einen Parameter <controlpar_i> ist als Eingabe erlaubt:

- PROTOCOL = Y oder PROTOCOL = N
- MESSAGE = Y oder MESSAGE = N
- COMMENT = Y oder COMMENT = N
- LIST = Y oder LIST = N
- LISTAGP = Y oder LISTAGP = N
- EQUATION = Y oder EQUATION = N
- MAX = <number>

Das Kommando setzt einen oder mehrere Kontrollparameter auf den bzw. die eingegebenen Werte. Um es zu beenden, muß nach der letzten Eingabe ein Strichpunkt angegeben werden. Die Bedeutung der Kontrollparameter:

PROTOCOL : schaltet das Protokoll ein bzw. aus (initialisiert mit Y, d.h. das Protokoll ist eingeschaltet)

MESSAGE : steuert, ob die zu einer Datei gehörenden Nachrichten beim Öffnen derselben gelistet werden oder nicht (initialisiert mit Y, d.h. Nachrichten werden gelistet)

COMMENT : steuert, ob Kommentare beim Listen und Drucken von Objekten ausgegeben werden oder nicht (initialisiert mit Y, d.h. Kommentare werden gelistet)

LIST : steuert das automatische Listen von Objekten im SPEC- und MAP-Editor (initialisiert mit Y, d.h. Objekte werden gelistet)

LISTAGP : steuert das Listen von automatisch erzeugten Teilen von Objekten (initialisiert mit Y, d.h. diese Teile werden gelistet)

EQUATION : bestimmt, ob der Interpretierer Gleichungen ausnutzen soll (initialisiert mit N, d.h. Gleichungen werden nicht ausgenutzt)

MAX : begrenzt die Zahl von ununterbrochenen Evaluationszyklen im Interpretierer (initialisiert mit 10)

Kommandoname: **CONTROL**? alias **C**?

Parameter : --

Dieses Kommando listet die Kontrollparameter und deren aktuelle Werte.

Kommandoname: **COPY**

Parameter : <fileid_1>
<objectid_1>
<fileid_2> | @
<objectid_2>

Dieses Kommando kopiert das Objekt <objectid_1> aus der Datei <fileid_1> in die durch den dritten Parameter spezifizierte Datei. Das Objekt wird dabei in <objectid_2> umbenannt. Hierfür muß USER ein Zugriffsrecht für die Datei <fileid_1> haben. Die Datei, in die kopiert werden soll, muß eine private Datei von USER sein, die nicht in der Umgebung einer anderen Datei steht. Außerdem muß der neue Name des Objekts von allen Objektnamen der Zielhierachie verschieden sein. Der OK-Status des neuen Objekt ist "ungeprüft".

Kommandoname: **COPYFILE** alias **COPYF**

Parameter : <fileid_1> | @
<fileid_2>

Dieses Kommando kopiert die durch den ersten Parameter spezifizierte Datei in die neue Datei <fileid_2>. Die zu kopierende Datei muß entweder USER selbst gehören, oder USER muß ein Zugriffsrecht besitzen und die Umgebung darf nur aus Systemdateien bestehen. Die Umgebung der alten Datei wird als Umgebung der neuen Datei übernommen.

Kommandoname: **DELETE** alias **DEL**

Parameter : <fileid> | @

Dieses Kommando löscht die spezifizierte Datei von USER, falls sie nicht in der Umgebung einer anderen Datei liegt. Das Kommando wird nur ausgeführt, wenn auf die Rückfrage des Systems mit "y" geantwortet wird.

Kommandoname: **DELETEMESSAGE** alias **DELM**

Parameter : <fileid> | @
<number_1>
:
.
<number_n> ;

Dieses Kommando löscht Nachrichten, die zu der spezifizierten Datei gehören. Welche Nachrichten zu entfernen sind, wird durch die Parameter <number_i> spezifiziert. Jede Nachricht, deren Kennungzahl gleich einem <number_i> ist, wird gelöscht. Die restlichen Nachrichten werden umnumeriert. Um das Kommando zu beenden, muß nach dem letzten Parameter ein Strichpunkt angegeben werden.

Kommandoname: **DIRECTORY** alias **DIR**

Parameter : <fileid> | <userid>.* | * | @ | RL
SPEC | MAP | MOD | PROG | ENR | INST | REP | *

Dieses Kommando gibt Auskunft, welche Objekte sich auf welchen Dateien befinden. Ist der erste Parameter RL, so darf der zweite Parameter nur Objekttypen spezifizieren, die auf dem Realisierungslevel existieren, d.h. man darf nur MOD, PROG, ENR, INST, REP oder * angeben. Es werden alle Objekte des entsprechenden Typs gelistet, für die USER auf dem Realisierungslevel Zugriffsrechte besitzt. Sonst werden alle Objekte aufgelistet, deren Typ dem zweiten Parameter entspricht(d.h. entweder nur Spezifikationen, oder nur Maps, oder beide Objektarten zusammen), und die auf der/den durch den ersten Parameter spezifizierten Datei/Dateien existieren. Ist der erste Parameter gleich <userid>.* , so werden alle Dateien des entsprechenden Benutzers berücksichtigt, für die USER Zugriffsrechte besitzt.

Kommandoname: **END**

Parameter : --

Dieses Kommando beendet eine SPESY-Sitzung. Alle Änderungen, die im Verlauf der Sitzung an Objekten und Dateien gemacht wurden, werden von der SPESY-Verwaltung abgespeichert. Für den Fall, daß der Benutzer ein Protokoll führt bzw. geführt hat, fragt das System, ob ein Ausdruck erwünscht ist und druckt das Protokoll auf die Eingabe von "Y" hin aus. Die Datei hat den Namen <userid>.PRINTOUT.<versionsnummer> und wird nach dem Ausdrucken gelöscht. Antwortet der Benutzer mit "N", steht die Datei unter dem Namen <userid>.PROTOCOL zur Verfügung. SPESY kann darauf nicht mehr zugreifen. Der Benutzer befindet sich anschließend auf der BS2000-Betriebssystemebene.

Kommandoname: **ENVIRONMENT** alias **ENV**

```
Parameter   : <fileid> | @
              <fileid_1>
                  :
                  .
              <fileid_n> ;
```

Dieses Kommando erlaubt die Manipulation der Umgebung einer Datei, die durch den ersten Parameter spezifiziert wird. Diese muß USER gehören. Die neue Umgebung wird durch die **transitive Hülle** der Dateien definiert, die durch die weiteren Parameter spezifiziert werden. Ein Strichpunkt muß nach dem letzten Parameter folgen. Die neue Umgebung darf nur aus Systemdateien und Dateien von USER gebildet sein. Alle Objekte der neuen Umgebung müssen korrekt und alle Objektnamen paarweise verschieden sein.
Die neu berechnete Umgebung wird mit der bisher gültigen verglichen. Dabei wird festgestellt, welche Dateien bezüglich der alten Umgebung neu und welche nicht mehr darin enthalten sind. Danach wird für jede neue Datei untersucht, ob es erlaubt ist, diese in die bisher gültige Umgebung aufzunehmen. Für jede nicht mehr enthaltene Datei wird entschieden, ob sie aus der gültigen Umgebung entfernt werden darf. Damit eine Datei in eine bestehende Umgebung aufgenommen werden kann, müssen die folgenden Bedingungen erfüllt sein. Es sind zwei Fälle zu unterscheiden:

a) die zu erweiternde Umgebung gehört zu einer Datei, die an der Spitze einer Dateihierachie steht. Aufnahmebedingungen:

   - es darf durch die Hinzunahme kein Zyklus in der Umgebung entstehen.
   - die Objekte der Datei müssen ok sein.

- die Objektnamen müssen sich von allen anderen in der bestehenden Umgebung unterscheiden.

b) die zu erweiternde Umgebung gehört zu einer Datei (DAT), die selbst einer Umgebung angehört. Aufnahmebedingungen:

- zusätzlich zu den Bedingungen aus a) ist zu beachten, daß die aufzunehmende Datei in der Umgebung jener Dateien enthalten ist, in deren Umgebung sich DAT befindet.

Damit eine Datei aus der bestehenden Umgebung entfernt werden kann, müssen folgende Bedingungen erfüllt sein:

- kein Objekt der Datei darf von einem anderen Objekt der Dateihierachie benutzt werden.
- die Umgebung ohne die zu entfernende Datei ist immer noch transitiv geschlossen.

Alle Dateien, für die obige Tests erfolgreich sind, werden zur bestehenden Umgebung hinzugefügt bzw. entfernt. Andernfalls werden entsprechende Fehlermeldungen ausgegeben.

Kommandoname: **ENVIRONMENT**? alias **ENV**?

Parameter : <fileid> | @

Dieses Kommando listet alle Dateien, die in der Umgebung der spezifizierten liegen. Die höchste Datei der Hierachie erscheint zuerst, dann die nächsttiefere usw. Für die spezifizierte Datei muß USER Zugriffsrechte besitzen.

Kommandoname: **FIND** alias **f**

Parameter : <objectid>

Dieses Kommando listet die Namen aller Dateien, die das spezifizierte Objekt enthalten, und für die USER Zugriffsrechte besitzt. Der Objekttyp wird für jede Datei mitgelistet.

Kommandoname: **HELP** alias **H** alias **?**

Parameter(optional!!) : <commandname>

Dieses Kommando listet alle Kommandos des Systemlevels und deren Abkürzungen, falls kein Parameter angegeben wurde. Ansonsten wird eine kurze Beschreibung des spezifizierten Kommandos auf den Bildschirm ausgegeben.

Kommandoname: **LISTFILES** alias **LISTF** alias **LF**

Parameter : <userid> | @ | *

Dieses Kommando listet alle Dateien des spezifizierten Benutzers oder aller Benutzer, für die USER Zugriffsrechte besitzt. Durch Eingabe von @ als Parameter werden die USER gehörenden aufgelistet.

Kommandoname: **MERGEFILE** alias **MF**

Parameter : <fileid_1> | @
<fileid_2> | @

Dieses Kommando kopiert die durch den ersten Parameter spezifizierte Quelldatei und alle Dateien deren Umgebung, die nicht in der Umgebung der durch den zweiten Parameter spezifizierten Zieldatei enthalten sind, in die Zieldatei. Die Zieldatei muß eine Datei von USER sein und an der Sitze eine Dateihierachie stehen. Alle Systemdateien der Umgebung der Quelldatei müssen auch in der Umgebung der Zieldatei sein. USER muß für alle Dateien, die kopiert werden, Zugriffsrechte besitzen. Die Objektnamen in den zu kopierenden Dateien müssen von allen Objektnamen in der Zieldatei und deren Umgebung verschieden sein. Sind alle genannten Bedingungen erfüllt, werden alle Objekte der Quellhierachie, die nicht in der Zielhierachie enthalten sind, in die Zieldatei kopiert.

Kommandoname: **MESSAGE** alias **M**

Parameter : <fileid> | @
<text>
SUPPRESS | NOSUPPRESS

Dieses Kommando fügt die durch <text> spezifizierte Nachricht in die Nachrichtenliste der spezifizierten Datei ein, falls USER MESSAGE-Recht für die Datei besitzt. Die Nachricht darf maximal 252 Zeichen lang sein, und muß in Anführungszeichen eingeschlossen sein. Treten im Text die Zeichen " und % auf, muß jeweils das Zeichen % vorangestellt werden. Die Nachrichtenliste wird automatisch umnumeriert, so daß die neue Nachricht die Nummer 1 erhält.
Der dritte Parameter wird nur vom System-Manager verlangt. Er kann damit bestimmen, ob seine Nachricht beim Ausführen des OPEN-Kommandos durch andere Benutzer unterdrückt werden kann oder nicht.

Kommandoname: **MESSAGE?** alias **M?**

Parameter : <fileid> | @

Dieses Kommando listet alle Nachrichten, die zu der spezifizierten Datei gehören, falls USER Zugriffsrechte für diese besitzt.

Kommandoname: **MOVE**

Parameter : <objectid>
<fileid_1>
<fileid_2>

Dieses Kommando verschiebt das Objekt <objectid> aus der Quelldatei <fileid_1> in die Zieldatei <fileid_2>, falls folgende Bedingungen zur Erhaltung der Konsistenz der Datei-hierachie und der Korrektheit des Objekts erfüllt sind:

- die Zieldatei muß in der Umgebung der Quelldatei enthalten sein.
- das Objekt muß ok sein.
- der Name des Objekts muß von allen Objektnamen der Dateien verschieden sein, die die Zieldatei in ihrer Umgebung enthalten.
- Objekte, die vom zu verschiebenden Objekt benutzt werden, müssen entweder in der Zieldatei selbst oder in deren Umgebung verfügbar sein.

Falls USER der System-Manager ist und er versucht, ein Objekt einer Datei eines anderen Benutzers in eine Systemdatei zu

verschieben, ist eine weitere Bedingung zu erfüllen:

- der Name des zu verschiebenden Objekts muß sich von den Objektnamen aller Systemdateien unterscheiden.

Das Objekt wird in die Zieldatei eingetragen und aus der Quelldatei entfernt.

Kommandoname: **OPEN** alias **O**

Parameter : <fileid> | @
<fileid_1>
:
<fileid_n> ;

Dieses Kommando verläßt den Systemlevel und betritt den Filelevel. Der Benutzer kann dadurch
1.) Objekte einer existierenden, ihm gehörenden Datei zugänglich machen, oder
2.) er kann eine neue Datei für sich anlegen.
Der erste Parameter bezeichnet die Datei, die geöffnet werden soll.

Existiert die spezifizierte Datei bereits, so darf es nur eine Datei sein, die an der Spitze einer Dateihierachie steht. Dadurch ist garantiert, daß Dateien, die korrekte Objekte enthalten und die in der Umgebung anderer Dateien enthalten sind, nicht verändert werden können. Außerdem darf die Datei nicht durch eine Beweiseraufgabe blockiert sein. Wenn eine Beweisaufgabe zu einem Objekt der spezifizierten Datei gelöst ist, wird das Ergebnis eingetragen und die Datei entblockiert. Das Kommando macht die spezifizierte Datei und alle Dateien der Umgebung, d.h. alle Objekte der Dateihierachie, zugänglich. Es wird nachgesehen, ob Nachrichten zu der Datei existieren, und diese werden in Abhängigkeit vom Wert des Kontrollparameters MESSAGE ausgegeben. Dann meldet sich der Filelevel.

Ist die spezifizierte Datei dem SPESY-System nicht bekannt, hat das Kommando zumindest einen weiteren Parameter. Für jede neue Datei muß nämlich eine Umgebung angegeben werden. Die Umgebung der neuen Datei berechnet sich aus der **transitiven Hülle** der Dateien, die durch die Parameter zwei bis n spezifiziert werden. Hinter dem letzten Parameter muß ein Strichpunkt folgen, um das Kommando abzuschließen. Die Umgebung darf nur aus System- und/oder Privatdateien von USER bestehen. Sie muß konsistent sein, d.h.
- alle Objekte müssen ok sein
- die Objektnamen müssen alle unterschiedlich sein.
Dann wird die Datei angelegt. Alle Benutzer erhalten MESSAGE-Recht. Es meldet sich der Filelevel.

FÜr den System-Manager hat das Kommando zur Zeit noch eine andere Bedeutung. Er darf den Filelevel nicht betreten, dafür aber in der vorliegenden Version zu Testzwecken den Realisierungslevel, damit er in diesem Subsystem als "Benutzer" arbeiten kann. Das Subsystem arbeitet auf einer Dateihierachie. Der System-Manager gibt also als Parameter eine Datei irgend eines Benutzers an, die an der Spitze einer Dateihierachie steht. Die Objekte der Hierachie werden zugänglich gemacht und es wird direkt in den Realisierungslevel verzweigt.

Kommandoname: **PRINT**

```
Parameter   : <fileid> | @
              SPEC | MAP | *
```

Dieses Kommando druckt alle Objekte des durch den zweiten Parameter spezifizierten Typs, die auf der spezifizierten Datei stehen. USER muß Zugriffsrecht auf die Datei haben. Für jedes Objekt wird ein Druckauftrag erzeugt. Dieser gibt das Objekt auf eine Datei aus, deren Namen aus dem Objektnamen generiert wird. Der Name der Datei endet mit PRINTFILE.<versionsnummer>.

Kommandoname: **RIGHT** alias **R**

```
Parameter   : <userid> | *
              <fileid> | @ | *
              MESSAGE | READ | NONE
```

Dieses Kommando erlaubt USER, die Zugriffsrechte auf seine Dateien zu verändern (d.h. Löschen, Einfügen und Umändern). Der erste Parameter spezifiziert, für welchen Benutzer bzw. daß für alle Benutzer die Zugriffsrechte verändert werden sollen. Der zweite gibt an, für welche Datei bzw. daß für alle Dateien von USER das Zugriffsrecht in das durch den dritten Parameter angegebene Recht umgewandelt werden soll. Existierende Rechte werden überschrieben, nur die des System-Managers können nicht verändert werden.

Kommandoname: **RIGHT?** alias **R?**

Parameter : --

Dieses Kommando listet für jede Datei von USER, welche anderen Benutzer Zugriffsrechte dafür besitzen und ob diese MESSAGE- oder READ-Recht haben.

4.4. Befehle des Filelevels

Filelevelkommandos, die Objekte verändern, beziehen sich immer nur auf solche Objekte, die in der geöffneten Datei definiert sind. Dadurch wird sichergestellt, daß die Umgebung einer Datei auf diesem Level nicht verändert werden kann.
Der Parameter <objectid> in einem Kommando bedeutet immer, daß entweder ein Specid oder ein Mapatom angegeben werden muß. Ein Mapatom hat die Form <sourceid>.<targetid)_<arrowid>. Die einzigen Ausnahmen bilden das INPUT- und das EDIT-Kommando für Maps. Dort wird als Eingabe ein **simple Mapterm** gefordert. Die Syntax eines solchen Mapterms ist in [Spa 85] erläutert.

4.4.1. Übersicht

CHECK				
CONTROL	alias	**C**		
CONTROL?	alias	**C?**		
DELETE	alias	**DEL**		
DELETEMESSAGE	alias	**DELM**		
DIRECTORY	alias	**DIR**		
EDIT				
EDOR				
END				
ENVIRONMENT	alias	**ENV**		
ENVIRONMENT?	alias	**ENV?**		
FIND	alias	**F**		
GENERATE				
HELP	alias	**H**	alias	**?**
INPUT				
LIST	alias	**L**		
LISTFILES	alias	**LISTF**	alias	**LF**
MESSAGE	alias	**M**		
MESSAGE?	alias	**M?**		
PRINT	alias	**P**		
PROVE				
RL				

4.4.2. Beschreibung der Kommandos

Kommandoname: **CHECK**

Parameter : <objectid> | @

Dieses Kommando überprüft das spezifizierte Objekt auf seine syntaktische Korrektheit und die Einhaltung der Kontextbedingungen. Es setzt alle OK-Flags auf den berechneten Wert, d.h. auf "korrekt" oder "nicht korrekt". Die Korrektheitbedingungen für Spezifikationen sind in [Lic 85] beschrieben, die für Maps in [Spa 85].

Kommandoname: **CONTROL** alias **C**

Siehe 4.3.2.2. Beschreibung des CONTROL-Kommandos auf dem Systemlevel.

Kommandoname: **CONTROL?** alias **C?**

Siehe 4.3.2.2. Beschreibung des CONTROL?-Kommandos auf dem Systemlevel.

Kommandoname: **DELETE** alias **DEL**

Parameter : <objectid> | @

Dieses Kommando löscht das spezifizierte Objekt aus der geöffneten Datei, falls USER auf die RUckfrage des Systems mit "Y" antwortet. Die entsprechenden OK-Flags von Objekten, die das gelöschte Objekt benutzten, werden auf "ungeprüft" gesetzt.

Kommandoname: **DELETEMESSAGE** alias **DELM**

Parameter : <number_1>
:
.
<number_n> ;

Dieses Kommando löscht Nachrichten, die zur geöffneten Datei gehören. Die Parameter <number_i> geben an, welche Nachrichten zu entfernen sind. Jede Nachricht, deren Kennungzahl gleich einem <number_i> ist, wird gelöscht. Die restlichen Nachrichten werden umnumeriert. Das Kommando muß durch einen Strichpunkt nach dem letzten Parameter beendet werden.

Kommandoname: **DIRECTORY** alias **DIR**

Siehe 4.3.2.2. Beschreibung des DIRECTORY-Kommandos auf dem Systemlevel.

Kommandoname: **EDOR**

Parameter : --

Dieses Kommando startet das Dateibearbeitungsprogramm **EDOR**. Man kann damit eine Datei erzeugen oder abändern. Eine Spezifikation, die von einer mit **EDOR** erstellten Datei eingelesen werden soll (siehe INPUT-Kommando), muß auf einer Datei vom Typ SAM stehen. Will man eine solche Datei neu erstellen gibt man die EDOR-Kommandofolge **OV,<filename>,S,N!B** ein. Sonst **OV,<filename>,S!B** um eine bereits existierende Datei zu verändern.

Hinweis: Dieses Kommando wird lediglich zu Testzwecken während der Entwicklungszeit von SPESY zur Verfügung gestellt!

Kommandoname: **EDIT**

Parameter : SPEC | MAP
<objectid>

Dieses Kommando verläßt den Filelevel und betritt den Spec-Editorlevel oder den Map-Editorlevel abhängig vom ersten Parameter. Der andere Parameter bezeichnet das Objekt, mit dem der Editor betreten wird. Es muß in der geöffneten Datei definiert sein, um es editieren zu können. Objekte, die in der Umgebung der geöffneten Datei existieren, sind nicht editierbar. Ist das angegebene Objekt neu, d.h. weder in der geöffneten Datei noch in deren Umgebung vorhanden, wird vom jeweiligen Editorlevel direkt in den entsprechenden Inputlevel verzweigt.

Kommandoname: **END**

Parameter : --

Dieses Kommando verläßt den Filelevel und kehrt zum Systemlevel zurück. Alle Änderungen, die auf dem Filelevel gemacht wurden, d.h. alle Veränderungen, die sich auf Objekte beziehen, werden abgespeichert.

Kommandoname: **ENVIRONMENT**? alias **ENV**?

Parameter : --

Dieses Kommando listet die Umgebung der geöffneten Datei. Die höchste Datei der Hierachie erscheint zuerst, dann die nächsttiefere usw.

Kommandoname: **FIND** alias **F**

Parameter : <identifier>

Dieses Kommando durchsucht die gesamte Dateihierachie, d.h. alle Objekte, der geöffneten Datei. Dabei wird nach folgenden Kriterien vorgegangen:

a) Es wird überprüft, ob <identifier> ein **ASPIK-Objekt** in der Hierachie ist. In diesem Fall wird diese Information und zusätzlich der Objekttyp ausgegeben.
b) Darüberhinaus wird überprüft, ob <identifier> ein **Sorten-** oder **Operationsname** einer **Sorts-, Ops-, Auxiliaries-** oder **Privateopsklausel** ist, d.h. nach jedem Auftreten von <identifier> wird gesucht. Eine entsprechende Meldung wird ausgegeben, falls die Suche erfolgreich war.

Kommandoname: **GENERATE**

Parameter : <specterm>
<name> | =

Dieses Kommando generiert aus dem Specterm <specterm> eine Spezifikation. Sie ist unter dem durch den zweiten Parameter spezifizierten Namen ansprechbar. Der Name der erzeugten Spezifikation ist frei wählbar, er muß sich nur von allen Objektnamen der Dateihierachie unterscheiden. Wird für den zweiten Parameter ein = angegeben, ist <specterm> der Name der Spezifikation.

Kommandoname: **HELP** alias **H** alias **?**

Parameter(optional!!) : <commandname>

Dieses Kommando listet alle Kommandos des Filelevels und deren Abkürzungen, falls kein Parameter angegeben wurde. Ansonsten wird eine kurze Beschreibung des spezifizierten Kommandos auf den Bildschirm ausgegeben.

Kommandoname: **INPUT**

Parameter : SPEC | MAP | SPECFROM | MAPFROM
<objectid> | <fileid>

Dieses Kommando verläßt den Filelevel und betritt - implizit über den entsprechenden Editorlevel - den Inputlevel für Spezifikationen oder Maps. Die vier Alternativen des ersten Parameters bedeuten:

- SPEC Betreten des Inputlevels für Spezifikationen, wobei die Eingabe im Dialog erfolgt. Der zweite Parameter muß entweder ein existierender Spezifikationsname sein, der in der geöffneten Datei definiert ist, oder ein neuer, korrekter Spezifikationsname.
- MAP Betreten des Inputlevels für Maps, wobei die Eingabe im Dialog erfolgt. Der zweite Parameter muß entweder ein existierender **simple Mapterm** [Spa 85] sein, der in der geöffneten Datei definiert ist, oder ein neuer, korrekter **simple Mapterm.**
- SPECFROM Betreten des Inputlevels für Spezifikationen, wobei die Eingabe von einer Datei vom Typ SAM gelesen wird, die durch den zweiten Parameter spezifiziert wird. Auf der Datei muß als erstes der Name der Spezifikation stehen.
- MAPFROM Betreten des Inputlevels für Maps, wobei die Eingabe von einer Datei vom Typ SAM gelesen wird, die durch den zweiten Parameter spezifiziert wird.

Kommandoname: **LIST** alias **L**

Parameter : <objectid> | @ | *
TOTAL | INTERFACE | USEREL | REFREL | OK

Dieses Kommando listet für das durch den ersten Parameter spezifizierte Objekt, bzw. für alle Objekte der Hierachie, die durch den zweiten Parameter bezeichnete Information. Dabei bedeuten:

- TOTAL Das Objekt wird gelistet. Abhängig vom Kontrollparameter COMMENT werden Kommentare mitausgegeben oder nicht.
- INTERFACE Das Interface des Objekts wird gelistet, falls dieses keinen Zyklus enthält. Ist das Interface nicht korrekt, enthält aber keinen Zyklus, wird zusätzlich eine Warnung ausgegeben.
- USEREL Die Use-Beziehungen des Objekts werden in ei-

ner Tabelle aufgezeigt. Die Matrix ist folgendermaßen zu lesen:

- Alle Objekte der Matrix sind numeriert, wobei Bool immer die Nummer 1 besitzt. In jeder Zeile steht erst die Nummer, dann das zugehörige Objekt. Die Spalten der Matrix sind nur durch die Nummer gekennzeichnet.
- Besteht eine Use-Relation zwischen dem Element einer Spalte und einem einer Zeile, dann ist der Kreuzungspunkt mit * markiert. Alle Markierungen liegen im linken unteren Dreieck, falls keine Zyklen auftreten.

- REFREL — Die Referiert-Beziehungen des Objekts werden wie die Use-Beziehungen in Tabellenform dargestellt.
- OK — Die OK-Flags des Objekts und deren Werte werden in einer Baumstruktur aufgelistet. Die Struktur berücksichtigt die semantischen Zusammenhänge der OK-Flags. Die Wurzel des Baums bzw. eines Teilbaums ist also nur dann ok, wenn alle darunterliegenden Knoten ok sind. Die Werte der Flags, die die Semantik des Objekts betreffen, sind nicht in die Baumstruktur eingebettet und werden gesondert gelistet.

Kommandoname: **LISTFILES** alias **LISTF** alias **LF**

Siehe 4.3.2.2. Beschreibung des LISTFILES-Kommandos auf dem Systemlevel.

Kommandoname: **MESSAGE** alias **M**

Parameter : <text>

Dieses Kommando fügt die durch <text> spezifizierte Nachricht in die Nachrichtenliste der geöffneten Datei ein. Die Nachricht darf maximal 252 Zeichen lang sein, und muß in Anführungszeichen eingeschlossen sein. Treten im Text die Zeichen " und % auf, muß jeweils das Zeichen % vorangestellt werden. Die Nachrichtenliste wird automatisch umnumeriert, so daß die neue Nachricht die Nummer 1 erhält.

Kommandoname: **MESSAGE?** alias **M?**

Parameter : --

Dieses Kommando listet die zur geöffneten Datei gehörenden Nachrichten. Abhängig vom Kontrollparameter MESSAGE werden nur die nichtunterdrückbaren oder alle Nachrichten ausgegeben.

Kommandoname: **PRINT** alias **P**

Parameter : <objectid> | @
TOTAL | USEREL | REFREL | OK

Dieses Kommando druckt für das spezifizierte Objekt die gewünschte Information aus. Dabei bedeutet:

- TOTAL Das Objekt wird auf eine Datei ausgedruckt. Es wird aus dem Objektnamen ein Dateiname erzeugt, der mit PRINTFILE.<versionsnummer> endet. Auf diese Datei gibt ein vom System generierter Druckjob das Objekt aus.
- USEREL Die Use-Beziehungen des Objekts werden auf die Datei USO.MOD.OUTFILE.<versionsnummer> in Tabellenform ausgegeben. Die Interpretation der Tabelle ist in 4.4.2. im LIST-Kommando beschrieben.
- REFREL Die Referiert-Beziehungen des Objekts werden wie die Use-Beziehungen in Tabellenform auf die Datei USO.MOD.OUTFILE.<versionsnummer> ausgegeben.
- OK Die OK-Flags des Objekts und deren Werte werden in der Darstellung des LIST-Kommandos in 4.4.2. auf die Datei OUTFILE.OK.STATUS.<versionsnummer> ausgegeben.

Kommandoname: **PROVE**

Parameter : <objectid> | @
USER | MKRP | RRLAB

Dieses Kommando erlaubt es, für das spezifizierte Objekt Beweisaufgaben zum Nachweis der Konsistenz, der Termination und der Herleitbarkeit zu generieren. Gibt man für den zweiten Parameter USER an, so kann der Benutzer in der vorliegenden Version zu Testzwecken die OK-Flags der genannten Bedingungen

im Dialog setzen, ohne einen automatischen Beweiser einzuschalten. Der Benutzer kann damit Ergebnisse manueller Beweise eingeben. Bei Eingabe von MKRP wird eine Beweisaufgabe für den Resolutionsbeweiser "Markgraf Karl Refutation Procedure" generiert. Die Datei wird bis zur Lösung der Aufgabe gesperrt, das System beendet automatisch den Filelevel und kehrt auf den Systemlevel zurück. Die Änderungen, die auf dem Filelevel durchgeführt wurden, werden abgespeichert. Mit diesem Beweiser kann die Termination der Operationsdefinitionen und die Herleitbarkeit der Properties der Objekte bewiesen werden. Zum Nachweis der Konsistenz von Properties ist RRLAB als Parameter anzugeben. Damit wird die Schnittstelle zum Rewrite-Rule-Labor generiert. Dort kann auch festgestellt werden, ob die Operationsdefinitionen des Objekts terminieren.

Kommandoname: **RL**

Parameter : --

Dieses Kommando verläßt den Filelevel und betritt den Realisierungslevel. Alle Objekte der Dateihierachie sind weiterhin zugreifbar.

4.5. Befehle des Editors für Spezifikationen und Maps

In jedem Editor erscheint nach dem Betreten und nach jeder Abarbeitung eines Befehles eine Kopfzeile, die folgende Informationen enthält:

1. Name des editierten Objekts
2. Die aktuelle Position des Editors.

Nach jedem Kommando wird das Objekt ab der momentanen Postition gelistet, falls der Kontrollparameter "LIST" mit "Y" besetzt ist. Ist der Kontrollparameter "COMMENT" ebenfalls mit "Y" besetzt, so werden auch die zugehörigen Kommentare gelistet.

Vor dem Aufruf eines der Änderungskommandos **DELETE**, **CHANGE** oder **INSERT** muß auf eine Klausel positioniert worden sein. Beim Ändern eines Klauselelementes kann gleichzeitig das zugehörige Kommentarklauselelement geändert werden. Die aktuelle Position wird durch die Änderungskommandos nicht verändert.

Am Ende einer Editorsitzung kann der Benutzer alle bis dahin durchgeführten Änderungen durch das ABORT-Kommando rückgängig machen oder durch das END-Kommando abspeichern.

Die im folgenden beschriebenen Kommandos gelten alle sowohl für den Editor der Spezifikationen als auch für den Editor der Maps.

4.5.1. Übersicht

ABORT	
CHECK	
CHANGE	alias **CH**
CONTROL	alias **C**
CONTROL?	alias **C?**
DATE	alias **D**
DELETE	alias **DEL**
END	
HELP	alias **?**
INPUT	
INSERT	alias **I**
LIST	alias **L**
<NODE-ID>	
<NUMBER>	
+	
-	

Bemerkung: Für **<NODE-ID>** können im Editor für Spezifikationen folgende Klauselnamen eingesetzt werden : **SPEC CMT USE SORTS OPS PROPS CONS AUX DEFAUX DEFCAR DEFCONS PRIVOPS DEFOPS.**

Im Editor für Maps entsprechend : **MAP IS USE BASE SORTS OPS.**

4.5.2. Beschreibung der Kommandos

Kommandoname: **ABORT**

Parameter : --

Mit diesem Kommando wird der Editor verlassen und zum Filelevel zurückgekehrt. Alle Änderungen, die im Editor bzw. im Inputsystem an dem jeweiligen Objekt vorgenommen wurden, werden vergessen. Das Objekt bleibt also unverändert erhalten.

Kommandoname: **CHECK**

Parameter : --

Dieses Kommando übergibt das editierte Objekt dem entsprechenden Check-System. Dieses überprüft das eingegebene Objekt syntaktisch und kontextsensitiv. Wird ein Fehler erkannt, so wird dieser dem Benutzer mitgeteilt.

Kommandoname: **CHANGE** alias **CH**

Parameter : <number>
<text> ;

Dieses Kommando ermöglicht es, das durch <number> angegebene Klauselelement durch <text> zu ersetzen. <text> muß dabei den Syntaxregeln des Klauselelementes ([Lic 85], [Spa 85]) genügen. Ist <text> leer, so erfolgt keine Änderung.

Kommandoname: **CONTROL** alias **C**

Siehe 4.3.2.2. Beschreibung des CONTROL-Kommandos auf dem Systemlevel.

Kommandoname: **CONTROL?** alias **C?**

Siehe 4.3.2.2. Beschreibung des CONTROL?-Kommandos auf dem Systemlevel.

Kommandoname: **DELETE** alias **DEL**

Parameter : <number> | <node-id>

Dieses Kommando löscht die gesamte durch <node-id> angegebene Klausel, bzw. das durch <number> spezifizierte Klauselelement der aktuellen Position.

Kommandoname: **END**

Parameter : --

Mit diesem Kommando wird der Editor verlassen und zum Filelevel zurückgekehrt. Es werden alle Änderungen, die im Editor bzw. im dazugehörenden Eingabesystem vorgenommen wurden, wirksam.

Kommandoname: **HELP** alias **?**

Parameter(optional!!) : <commandname>

Falls der Parameter <commandname> nicht angegeben wird, werden alle Editorkommandos gelistet. Sonst erhält man eine genaue Beschreibung des durch <commandname> angegebenen Kommandos.

Kommandoname: **INPUT**

Parameter : --

Es erfolgt eine Verzeigung zum jeweiligen Eingabesystem. Dort kann die Eingabe von Klauseln fortgesetzt werden, falls das System einen korrekten Aufsetzpunkt findet (siehe 4.6.2. und 4.7.2.).

Kommandoname: **INSERT** alias **I**

Parameter : <number>
<text> ;

Dieses Kommando fügt <text>, der den Syntaxregeln entsprechen muß, vor die durch <number> angegebene Position ein. Falls <number> die Anzahl der Klauselelemente übersteigt, wird der <text> als letztes Klauselelement eingefügt. Ist <number> kleiner als eins, wird <text> vor das erste Klauselelement eingefügt.

Kommandoname: **LIST** alias **L**

Parameter : --

Es wird ein Listing der aktuellen Position erzeugt.

Kommandoname: **<NODE-ID>**

Parameter : --

Durch die Angabe eines KLauselnamens (siehe Bemerkung in 4.5.1.) positioniert der Editor auf diese gesamte Klausel.

Kommandoname: **<NUMBER>**

Parameter : --

Durch die Angabe einer ganzen positiven Zahl wird, abhängig von der positionierten Klausel, auf das durch die Zahl angegebene Klauselelement positioniert.

Kommandoname: +

Parameter : <number> | +

Dieses Kommando schiebt das Listing um <number> Zeilen vor. Wird als Parameter '+' angegeben, so wird die letzte Seite des Listings angezeigt. Ist der Kontrollparameter "LIST" nicht auf automatisches Listen gesetzt, wird ein Listing erstellt.

Kommandoname: -

Parameter : <number> | -

Dieses Kommando schiebt das Listing um <number> Zeilen zurück. Falls als Parameter '-' angegeben wird, wird der Anfang des Listings angezeigt. Ist der Kontrollparameter "LIST" nicht auf automatisches Listen gesetzt, wird ein Listing erstellt.

4.6. Benutzeranleitung für die Eingabe von Spezifikationen

4.6.1. Allgemeine Beschreibung

Das Eingabesystem für Spezifikationen (Spec-Inputlevel) ermöglicht die interaktive und syntaxorientierte Definition einer neuen ASPIK-Spezifikation oder die Erweiterung einer schon bestehenden, solange sie sich auf einer Privatdatei befindet. Während der Eingabe wird die interne Darstellung der Spezifikation in Form eines Syntaxbaumes (siehe Abb.4) aufgebaut. Der Anfangszustand bei der Eingabe einer neuen Spezifikation ist durch den leeren Syntaxbaum gekennzeichnet. Bei der Erweiterung einer alten Spezifikation läßt sich der Syntaxbaum in zwei Teile zerlegen, in einen vollständig gefüllten, kontextfrei und kontextsensitv korrekten linken Teil sowie in einen leeren rechten Teil. Diese Vorbedingung muß am Anfang, während und am Ende der Eingabe erfüllt sein, um kontextsensitive Korrektheitseigenschaften garantieren zu können. Kontextsensitive Tests finden schritthaltend mit der Eingabe statt und erlauben eine frühestmögliche Fehlererkennung. Im Fehlerfall wird die Eingabe zurückgesetzt und eine Korrektur ermöglicht.

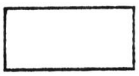
Wenn das Eingabesystem aktiviert wird und alle Vorbedingungen erfüllt sind, wird auf den nächsten Hauptknoten im Syntaxbaum positioniert, der noch nicht vollständig gefüllt ist. Kann das System in einer bereits existierenden Spezifikation keinen korrekten Aufsetzpunkt finden, weil diese beispielsweise nicht korrekte Klauseln enthält, wird eine Fehlermeldung ausgegeben und auf den Editor für Spezifikationen zurückgesprungen. Im ersten Fall, d.h. eine korrekte Startklausel wurde gefunden, fordert das System eine Eingabe an. Kann diese automatisch generiert werden, erzeugt SPESY diese Eingabe und fordert weitere Eingaben an. Diese systemgenerierten Teile einer Spezifikation sind in Abb.5 speziell gekennzeichnet. So kann eine ganze Klausel generiert werden (z.B. define-carriers-clause), ein Klauselelement oder auch nur ein Klauselschlüsselwort (z.B. ops-clause, define-ops-clause). Der Benutzer kann seine Eingabe bis zu einer beliebigen, sogenannten Promptposition fortsetzen, d.h. bis zur nächsten oder im Extremfall bis zur letzten Promptposition im Syntaxbaum. Die Promptpositionen sind in der Abbildung 5 durch **PP** dargestellt. Dieses Konzept erlaubt es dem mit ASPIK vertrauten Benutzer, seine Spezifikationen an einem Stück einzugeben, während der unerfahrene Benutzer durch die Syntax 'gelotst' wird.

Zusätzlich ist es möglich, eine Spezifikation von einer Datei einzulesen. Tritt hierbei ein Fehler auf oder wird das Ende der Spezifikation erkannt, so schaltet SPESY auf Eingabe vom Bildschirm um und beendet das Einlesen von der Datei. Falls das System in der Benutzereingabe keinen Fehler entdeckt, kann die Eingabe wie vorher beschrieben fortgesetzt werden. Ansonsten wird eine Fehlermeldung ausgegeben, aus der die Fehlerursache und die Position, ab der die Eingabe eventuell ignoriert wurde, hervorgeht. Gleichzeitig wird die Eingabe an die letzte Promptposition der Hauptknoten zurückgesetzt, so daß

der Benutzer den Fehler korrigieren kann.

Nach jedem Klauselelement kann der Benutzer Kommentare eingeben, die mit /* beginnen und mit */ enden müssen.

In Abb.5 wird folgende Notation verwendet:

 kennzeichnet die automatisch generierbaren Teile einer Spezifikation.

GROSSCHREIBUNG kennzeichnet die Terminalsymbole.

:: trennt linke und rechte Seite einer Regel.

() sind Meta-Klammern.

[] umschließen optionale Teile.

PP kennzeichnet die Promptpositionen an den Hauptknoten.

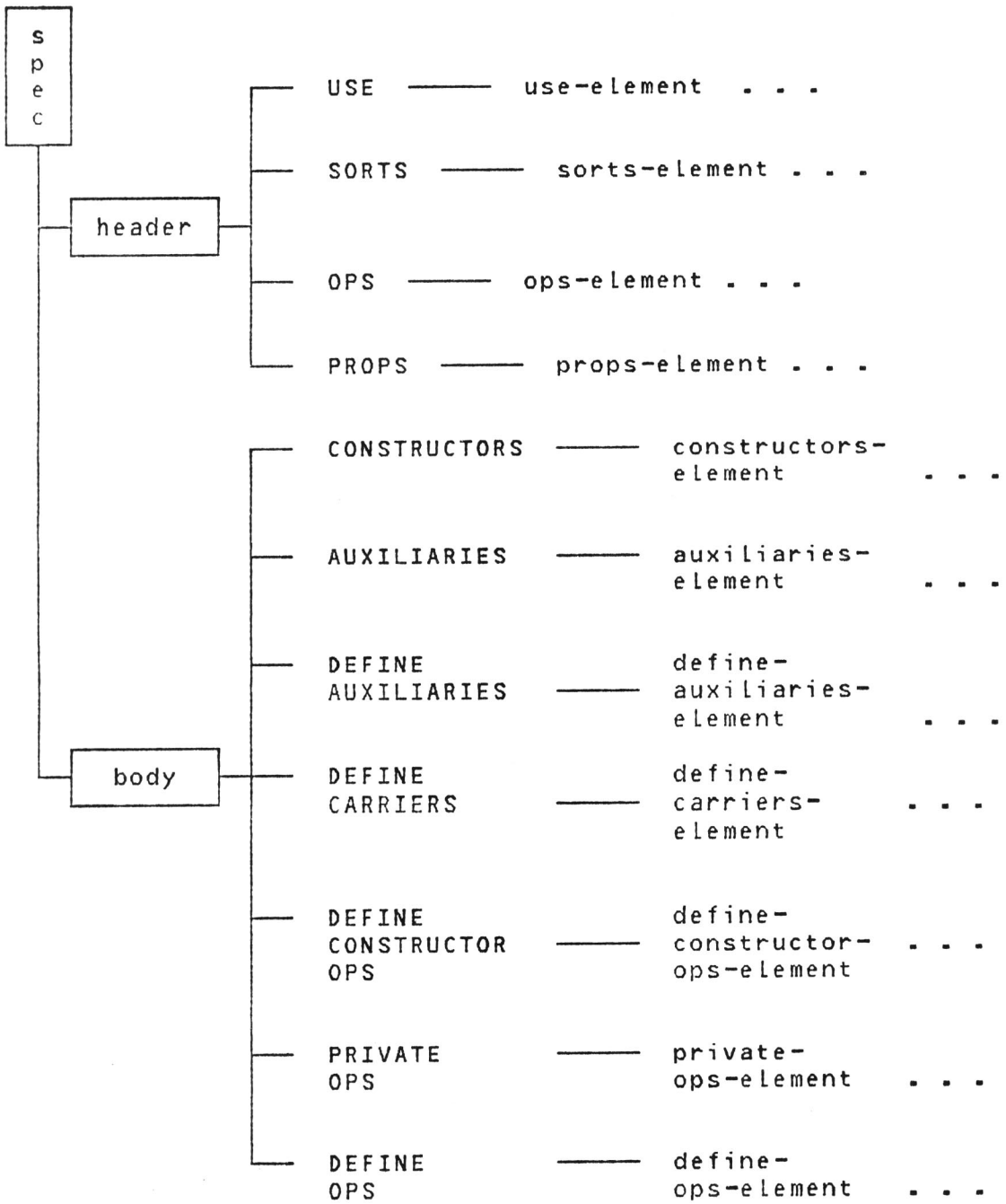


Abb.4 : Struktur des Syntaxbaumes einer Spezifikation. Hauptknoten (Klauseln und Klauselelemente) sind alle nicht umrahmten Knoten.

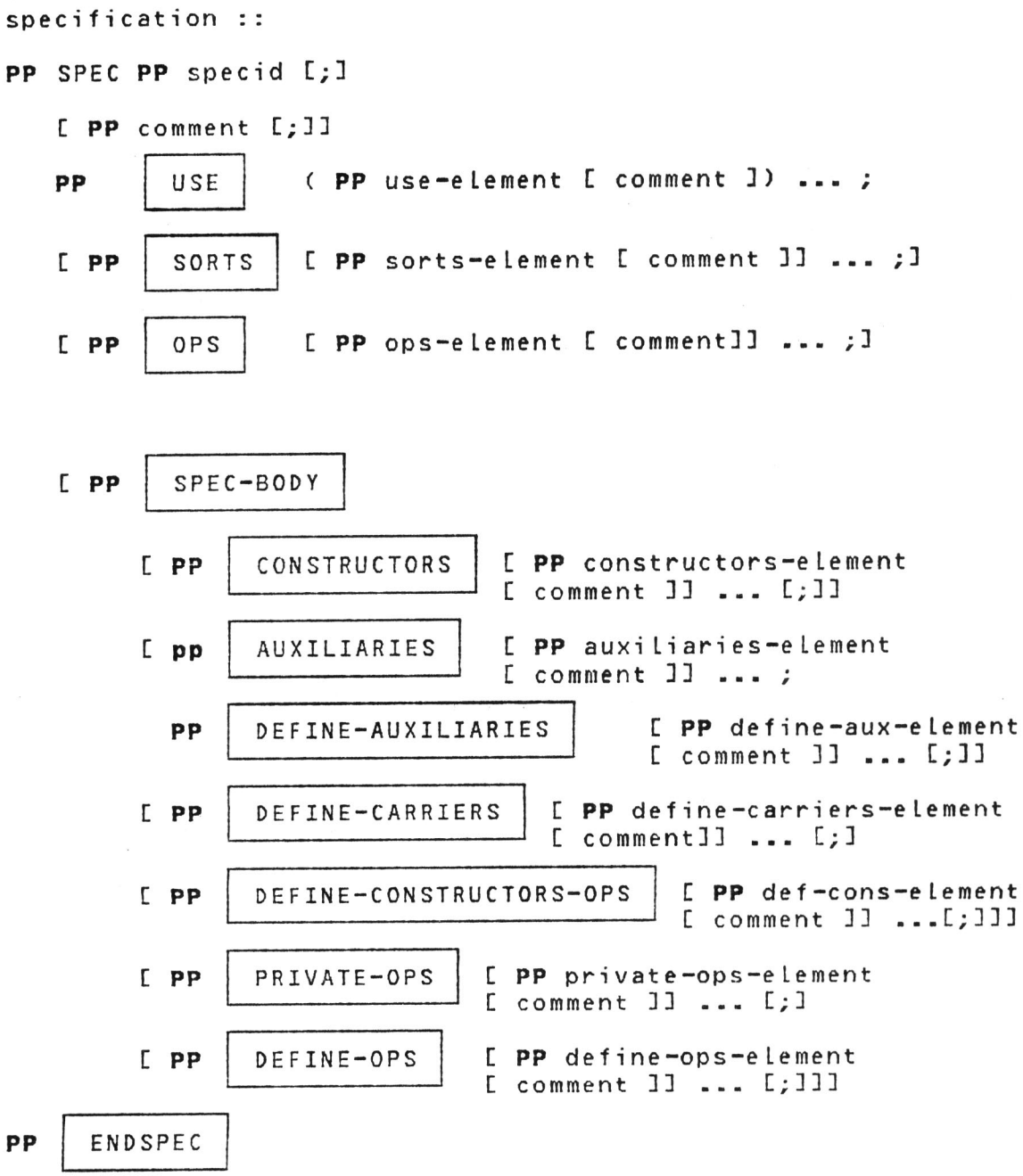


Abb.5 : Syntax einer Spezifikation bis zu den Klauselelementen ohne Berücksichtigung der Leerzeichen und Kommata.

Dem Benutzer stehen an jeder Promptposition außer an den Promptpositionen in den Operationsschemata der define-auxiliaries-clause und der define-ops-clause zwei verschiedene Alternativen zur Fortführung des Dialogs zur Verfügung:

1. Die Eingabe fortsetzen
2. Kommandos angeben.

4.6.2. Die Kommandos des Spec-Inputlevels

HELP alias **H** alias **?**
END

Beschreibung der Kommandos

Kommandoname: **END**

Parameter : --

Dieses Kommando beendet den Eingabedialog. Danach befindet sich der Benutzer im Spec-Editorlevel, bzw. auf dem Filelevel, falls die Spezifikation vollständig ist.

Kommandoname: **HELP** alias **H** alias **?**

Parameter : --

Dieses Kommando gibt dem Benutzer an jeder Promptposition Auskünfte über die an dieser Stelle erwartete Syntax. Die Informationen sind mehrstufig aufgebaut, d.h. ist die angegebene Information zu allgemein, so liefert ein erneutes HELP-Kommando detailliertere Auskünfte zur aktuellen Promptposition.

Aufruf

Der Aufruf des Eingabesystems für Spezifikationen erfolgt mit einem der folgenden Kommandos:

vom **Filelevel** :

EDIT SPEC <specid>, falls die Spezifikation neu ist
INPUT SPEC <specid>
INPUT SPECFROM <filename>, falls die Spezifikation auf der SAM-Datei <filename> steht;

vom **Spec-Editorlevel** :

INPUT, falls die editierte Spezifikation mit Systemunterstützung erweitert werden soll.

Eine weitergehende Beschreibung des Eingabesystems und Eingabedialogs für Spezifikationen ist in [Lic 85] zu finden.

4.7. Benutzeranleitung für die Eingabe von Maps

4.7.1. Allgemeine Beschreibung

Das Eingabesystem für Maps (Map-Inputlevel) ermöglicht die interaktive und syntaxorientierte Definition einer neuen ASPIK-Map oder die Erweiterung einer schon bestehenden. Während der Eingabe wird die interne Darstellung der Map in Form eines Syntaxbaumes (siehe Abb.6) aufgebaut. Der Anfangszustand bei der Eingabe einer neuen Map ist durch den leeren Syntaxbaum gekennzeichnet. Bei der Erweiterung einer alten Map läßt sich der Syntaxbaum in zwei Teile zerlegen, in einen vollständig gefüllten, kontextfrei und und kontextsensitv korrekten linken Teil sowie in einen leeren rechten Teil. Diese Vorbedingung muß am Anfang, während und am Ende der Eingabe erfüllt sein, um kontextsensitive Korrektheitseigenschaften garantieren zu können. Findet das Eingabesystem keinen korrekten Aufsetzpunkt, weil z.B. in der Map eine nicht korrekte Klausel enthalten ist, wird eine entsprechende Fehlermeldung ausgegeben und auf den Editor für Maps zurückgesprungen. Sonst finden syntaktische und kontextsensitive semantische Tests schritthaltend mit der Eingabe statt. Die jeweils erkannten Fehler werden angezeigt. Wird in der letzten Eingabe ein Fehler erkannt, wird die nicht mehr akzeptierte Eingabe und eine Fehlermeldung ausgegeben. Der Benutzer kann anstelle des abgewiesenen Eingaberestes eine neue Eingabe eingeben.

Während der Eingabe hat der Benutzer an bestimmten Punkten im Syntaxbaum, den sogenannten Promptpositionen (PP), die Möglichkeit, die Eingabe weiterzuführen, zu beenden oder sich über die momentan erwartete Syntax zu informieren (Abb.7). Beendet der Benutzer die Eingabe, ohne die Map vollständig eingegeben zu haben, wird auf den Map-Editorlevel zurückgekehrt. Nur wenn eine Map vollständig und korrekt eingegeben wurde, wird zum Filelevel zurückgekehrt.

Mit Hilfe der automatisch generierbaren Teile (siehe Abb.7) wird der Benutzer durch die Syntax 'gelotst'. Dadurch weiß der Benutzer, in welcher Klausel er sich befindet. Diese Teile werden dann generiert, wenn der Benutzer die Eingabe davor unterbricht. Ansonsten werden keine besonderen Eingabeaufforderungen ausgegeben. Einem mit ASPIK vertrauten Benutzer ist es daher möglich, eine Map an einem Stück einzugeben.

Nach einigen Klauselelementen kann der Benutzer Kommentare eingeben, die mit /* beginnen und mit */ enden müssen.

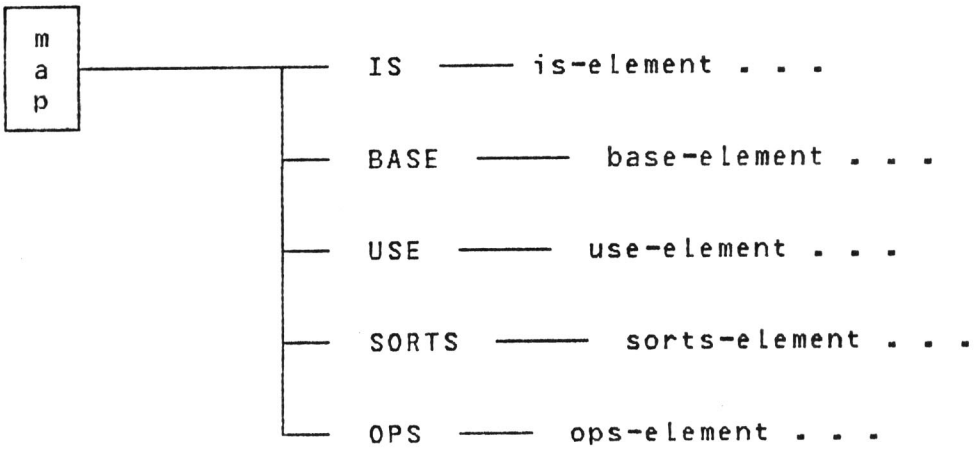


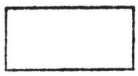
Abb.6 : Struktur des Syntaxbaumes einer Map. Hauptknoten (Klauseln und Klauselelemente) sind alle nicht umrahmten Knoten.

```
map ::
PP MAP PP mapid [;]
   [ PP comment [;]]
   PP    IS      what-map [ comment ] [;]
   [ PP  BASE    [ PP specterm [ comment ] ] ... ;]
   [ PP  USE     [ PP mapterm [ comment ]] ... ;]
   [ PP  SORTS   [ PP simple-sortid = sortid ] ... [;]]
   [ PP  OPS     [ PP simple-opid = opid ] ... [;]]

PP  ENDMAP
```

Abb.7 : Syntax einer Map bis zu den Klauselelementen ohne ohne Berücksichtigung der Leerzeichen und Kommata.

In Abb.7 wird folgende Notation verwendet:



kennzeichnet die automatisch generierbaren Teile einer Map.

GROSSCHREIBUNG kennzeichnet die Terminalsymbole.

:: trennt linke und rechte Seite einer Regel.

[] umschließen optionale Teile.

Dem Benutzer stehen an jeder Promptposition zwei Möglichkeiten zur Fortführung des Dialogs zur Verfügung:

1. Die Eingabe fortsetzen
2. Kommandos angeben.

Es ist zu beachten, daß bei auftretenden Fehlern das System die Eingabe nicht unbedingt bis zur letzten Promptposition zurücksetzt. Deshalb kann nicht nach jeder Fehlermeldung ein Kommando eingegeben werden. Eine leere Eingabe (Blanks) bewirkt jedoch ein Zurücksetzten zum letzten Fehleraufsetzpunkt. Über die letzte Promptposition kann nicht zurückgesetzt werden.

4.7.2. Die Kommandos des Map-Inputlevels

HELP alias **H** alias **?**
END

Beschreibung der Kommandos

Kommandoname: **END**

Parameter : --

Dieses Kommando beendet den Eingabedialog. Danach befindet sich der Benutzer im Map-Editorlevel, bzw. auf dem Filelevel, falls die Map vollständig ist.

Kommandoname: **HELP** alias **H** alias **?**

Parameter : --

Dieses Kommando gibt dem Benutzer an jeder Promptposition beim ersten Aufruf Auskunft Über die an dieser Stelle erwartete Syntax. Beim zweiten Aufruf werden die möglichen Kommandos gelistet und ein erneutes HELP-Kommando liefert eine Kurzbeschreibung aller Kommandos.

Aufruf

Der Aufruf des Eingabesystems für Maps erfolgt mit einem der folgenden Kommandos:

vom **Filelevel** :

EDIT MAP <simple Mapterm>, falls die Map neu ist
INPUT MAP <simple Mapterm>

vom **Map-Editorlevel** :

INPUT, falls die editierte Map mit Systemunterstützung erweitert werden soll.

Eine ausführliche Beschreibung der Syntax von Maps und deren kontextsensitiven Korrektheitsbedingungen ist in [Spa 85] zu finden.

4.8. Verhalten bei Fehlern

Sollte einmal ein Fehler auftreten, so übernimmt das INTERLISP-System die weitere Steuerung. Auf dem Bildschirm erscheint eine Fehlermeldung und das Prompt-Symbol:

```
1:
```

Hier muß der Benutzer folgendes Kommando eingeben:

```
(spesy.error)
```

Für den Fall, daß der Benutzer beim Auftreten des Fehlers ein Protokoll führt, wird dieses beendet. In diesem Fall, oder wenn während der Sitzung protokolliert wurde, fragt das System, ob ein Ausdruck erwünscht ist und druckt das Protokoll auf die Eingabe von "Y" hin aus. Die Datei hat den Namen <userid>.PRINTOUT.<versionsnummer> und wird nach dem Ausdrucken gelöscht. Antwortet der Benutzer mit "N", steht die Datei unter dem Namen <userid>.PROTOCOL zur Verfügung. SPESY kann darauf nicht mehr zugreifen. Der Benutzer befindet sich anschließend auf der BS2000-Betriebssystemebene.

Sollte es beim Aufruf von /DO SPESY zu Fehlern kommen, kann der Grund dafür nur in einer nicht vollständigen Abarbeitung des SPESY-Bootstraps beim davor liegenden SPESY-Aufruf, z.B. durch einen Maschinenfehler, zu finden sein. In diesem Fall muß das System durch

```
/DO SPESY.FAIL
```

initialisiert werden. Ein anschließender Aufruf von /DO SPESY führt dann zum gewünschten Ergebnis.

5. Beispielsitzungen

5.1. Sitzungsprotokoll System-Manager

Die folgende SPESY-Sitzung führte der System-Manager durch. Der Aufruf des SPESY-Systems aus der Betriebssystemebene und die Eingabe seiner Benutzerkennung SYS und seines Passwortes ist nicht protokolliert, da das Protokoll erst mit Betreten des SYSTEMLEVELS aktiviert wird. Zuerst informiert sich der System-Manager mit dem listfiles Kommando über alle im System existierenden Dateien . Dann informiert ihn das user? Kommando über alle registrierten Benutzer(kennungen) und die dazugehörigen Passwörter. Anschließend installiert er die neue Benutzerkennung TESTUSER mit dem Passwort BEISPIEL. Er erteilt dem neuen Benutzer MESSAGE-Recht für die Datei SYS.BOOLFILE und löscht die Benutzerkennung TESTUSER (und eventuell vorhandene Dateien) wieder. Durch das Kommando env? informiert sich SYS über die Dateihierachie von VS.TESTFILE1 des Benutzers VS. Das Kommando convert nimmt die aus der Privatdatei VS.TESTFILE generierte Datei SYS.CONVERT-TEST in die System-Bibliothek auf und löscht die Quelldatei. Das anschließende listfiles (lf) Kommando mit dem Dialogparameter @ bezieht sich auf die Dateien des aktuellen Benutzers. Es zeigt die konvertierte Datei als neue Systemdatei CONVERT-TEST. Ein erneutes environment? Kommando zeigt, daß die Dateihierachie von VS.TESTFILE1 jetzt die neue Systemdatei enthält.

```
(OUT)    PROTOCOL STARTED.
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      listfiles
(OUT)    ENTER USERNAME, * OR @ :
(OUT)
(IN)      *
(OUT)    YOU OWN MESSAGERIGHT FOR :
(OUT)       BOOLFILE
(OUT)       SBQ.SBQ-LEVEL1
(OUT)       SBQ.SBQ-LEVEL2
(OUT)       SBQ.SBQ-LEVEL3
(OUT)       SBQ.SBQ-LEVEL4
(OUT)       SBQ.CONTAINER--FILE
(OUT)       SBQ.STACK--FILE
(OUT)       VS.TESTFILE
(OUT)       VS.TESTFILE1
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      user?
(OUT)    USER        PASSWORD
(OUT)    SYS         XXXXXXX
(OUT)    VS          XXXXXXX
(OUT)    SBQ         XXXXXXX
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
```

```
(IN)      user
(OUT)    ENTER USERID:
(OUT)
(IN)      testuser
(OUT)    ENTER PASSWORD:
(OUT)
(IN)      beispiel
(OUT)    %  D800 ERASE FILE LISP.DATA.SPESY.SYSTEMFILE.00
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      user?
(OUT)    USER        PASSWORD
(OUT)    SYS         XXXXXXX
(OUT)    VS          XXXXXXX
(OUT)    SBQ         XXXXXXX
(OUT)    TESTUSER    BEISPIEL
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      right
(OUT)    ENTER NAME OF USER OR  *  :
(OUT)
(IN)      testuser
(OUT)    ENTER FILENAME ,  *  OR  @
(OUT)
(IN)      boolfile
(OUT)    ENTER RIGHT:  M E S S A G E , R E A D  OR  N O N E
(OUT)
(IN)      message
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      right?
(OUT)    FOR BOOLFILE:
(OUT)       USER        RIGHT
(OUT)       ^^^^^^^^^^^^^^^^^^^
(OUT)
(OUT)       SYS         MESSAGE
(OUT)       VS          READ
(OUT)       SBQ         READ
(OUT)       TESTUSER    MESSAGE
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      deleteuser testuser
(OUT)    %  D800 ERASE FILE LISP.DATA.SPESY.SYSTEMFILE.00
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      env? vs.testfile1
(OUT)
(OUT)    ALL ENVIRONMENTFILES:
(OUT)       VS.TESTFILE
(OUT)       BOOLFILE
```

```
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      convert
(OUT)    ENTER NAME OF SOURCEFILE :
(OUT)
(IN)      vs.testfile
(OUT)    ENTER NAME OF OBJECTFILE :
(OUT)
(IN)      convert-test
(OUT)    FILE CONVERT-TEST GENERATED !
(OUT)    %  D800 ERASE FILE LISP.DATA.VS.TESTFILE.00
(OUT)    SOURCEFILE DELETED
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      lf @
(OUT)    YOUR FILES :
(OUT)         CONVERT-TEST
(OUT)         BOOLFILE
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      env? vs.testfile1
(OUT)
(OUT)    ALL ENVIRONMENTFILES:
(OUT)       CONVERT-TEST
(OUT)       BOOLFILE
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      end
(OUT)    %  D800 ERASE FILE LISP.DATA.SPESY.SYSTEMFILE.00
```

5.2. Sitzungsprotokoll vom Systemlevel

Das folgende Protokoll zeigt den Beginn einer SPESY-Sitzung. Nach dem Aufruf von SPESY aus der Betriebssystemebene und der Eingabe seiner Benutzerkennung und seines Passwortes betritt der Benutzer SBQ den SYSTEMLEVEL. Die Sitzung wird defaultmäßig protokolliert. Das listfiles Kommando listet alle zugriffsberechtigten Dateien. Seine eigenen Dateien erscheinen ohne Präfix, während Dateien anderer Benutzer mit deren Benutzerkennung gepräfixt sind. Beispielsweise ist SBQ der Eigentümer von STACK--FILE und von CONTAINER--FILE; SYS.BOOLFILE ist die einzige Systemdatei. Das environment? Kommando zeigt, daß STACK--FILE über CONTAINER--FILE und dieser über SYS.BOOLFILE steht, den Regeln von Dateihierachien in SPESY entsprechend. Durch das directory Kommando erfährt SBQ, daß auf der Datei CONTAINER--FILE die Spezifikationen ELEM und LIMITED-LIFO und bisher auf STACK--FILE die Spezifikationen NAT, LIMIT und LIMITED-STACK abgelegt sind.

```
(OUT)    PROTOCOL STARTED.
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      listfiles *
(OUT)    YOU OWN MESSAGERIGHT FOR :
(OUT)       SBQ-LEVEL1
(OUT)       SBQ-LEVEL2
(OUT)       SBQ-LEVEL3
(OUT)       RLBASE
(OUT)       SBQ-LEVEL4
(OUT)       COST
(OUT)       AV.SORT
(OUT)       COST2
(OUT)       CONTAINER--FILE
(OUT)       STACK--FILE
(OUT)
(OUT)    YOU OWN READRIGHT FOR :
(OUT)       SYS.BOOLFILE
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      environment? stack--file
(OUT)
(OUT)    ALL ENVIRONMENTFILES:
(OUT)       CONTAINER--FILE
(OUT)       SYS.BOOLFILE
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      directory stack--file *
(OUT)    SPECS ON STACK--FILE      :   NAT   LIMIT   LIMITED-STACK
(OUT)    MAPS ON STACK--FILE      :THERE ARE NO MAPS!
(OUT)
```

```
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      directory container--file *
(OUT)    SPECS ON CONTAINER--FILE  :   ELEM   LIMITED-LIFO
(OUT)    MAPS ON CONTAINER--FILE  :THERE ARE NO MAPS!
(OUT)
(OUT)    SYSTEMLEVEL:
(OUT)
```

5.3. Sitzungsprotokoll vom Filelevel

Die Fortsetzung der Sitzung aus 5.2. wird protokolliert. Der Benutzer SBQ betritt den FILELEVEL, indem er STACK--FILE öffnet. Eine automatisch gelistete Meldung warnt ihn über eine unnötig deklarierte Funktion. Der Aufruf des list Kommandos listet die folgenden Informationen über die Spezifikation LIMITED-STACK:

- den Kontrollstrukturbaum, der anzeigt, daß der HEADER korrekt, der BODY aber wegen der falschen define-ops-Klausel nicht korrekt ist,
- die Matrix der USE-Beziehungen der Spezifikation, woraus hervorgeht, daß sie die Spezifikationen ELEM, LIMIT und NAT benutzt,
- das exportierte Interface,
- das Listing der Spezifikation, in der die undefinierte Operation bottom enthalten ist.

```
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open stack--file
(OUT)    FILE OPENED !
(OUT)    MESSAGE(S) FOR STACK--FILE :
(OUT)    1 : BOTTOM OPERATION IN LIMITED-STACK UNNECCESSARY
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-stack ok
(OUT)             status of specification:   LIMITED-STACK
(OUT)
(OUT)                      syntax: -                    semantics
(OUT)                                                       |
(OUT)                                                       |— terminating: ?
(OUT)                                                       |— consistent : ?
(OUT)          |                                            |— derivable  : ?
(OUT)       interface: +                  rest: -
(OUT)          |                             |
(OUT)          |                             |
(OUT)          |                             |
(OUT)          |                             |
(OUT)         use:   +          |           |              |
(OUT)         sorts: +    props: 0+  constructors: +   def.constructors: +
(OUT)         ops:   +               auxiliaries : +   private ops     : 0+
(OUT)                                  def.-aux.   : +   define ops      : -
(OUT)                                  def.-carrier: +
(OUT)
(OUT)              0 = empty
(OUT)              + = ok
(OUT)              - = not ok
(OUT)              ? = unknown
(OUT)    FILELEVEL:
(OUT)
```

```
(IN)     list limited-stack userel
(OUT)
(OUT)
(OUT)    spec LIMITED-STACK :  all used specification
(OUT)
(OUT)
(OUT)                          1  2  3  4  5
(OUT)     1          BOOL *
(OUT)     2           NAT *  *
(OUT)     3          ELEM *     *
(OUT)     4         LIMIT *  *     *
(OUT)     5 LIMITED-STACK *  *  *  *  *
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)     list limited-stack interface
(OUT)
(OUT)    ** THE EXPORTED INTERFACE OF  LIMITED-STACK  **
(OUT)
(OUT)    sorts from BOOL :
(OUT)       BOOL
(OUT)    operations from BOOL :
(OUT)       TRUE:  --> BOOL
(OUT)       FALSE:  --> BOOL
(OUT)       NOT: BOOL --> BOOL
(OUT)       _AND_: BOOL BOOL --> BOOL
(OUT)       _OR_: BOOL BOOL --> BOOL
(OUT)       EQ-BOOL: BOOL BOOL --> BOOL
(OUT)
(OUT)    sorts from NAT :
(OUT)       NAT
(OUT)    operations from NAT :
(OUT)       ZERO:  --> NAT
(OUT)       SUC: NAT --> NAT
(OUT)       PRED: NAT --> NAT
(OUT)       MULT: NAT NAT --> NAT
(OUT)       EXP: NAT NAT --> NAT
(OUT)       _+_: NAT NAT --> NAT
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)       _-_: NAT NAT --> NAT
(OUT)       _/_: NAT NAT --> NAT
(OUT)       ZERO?: NAT --> BOOL
(OUT)       _LT_: NAT NAT --> BOOL
(OUT)       _LE_: NAT NAT --> BOOL
(OUT)       _GT_: NAT NAT --> BOOL
(OUT)       _GE_: NAT NAT --> BOOL
(OUT)       EQ-NAT: NAT NAT --> BOOL
(OUT)
(OUT)    no sorts from LIMIT
(OUT)    operations from LIMIT :
(OUT)       LIMIT:  --> NAT
(OUT)
```

```
(OUT)    sorts from ELEM :
(OUT)       ELEM
(OUT)    no operations from ELEM
(OUT)
(OUT)    sorts from LIMITED-STACK :
(OUT)       STACK
(OUT)    operations from LIMITED-STACK :
(OUT)       EMPTY:  --> STACK
(OUT)       EMPTY?: STACK --> BOOL
(OUT)       FULL?: STACK --> BOOL
(OUT)       PUSH: STACK ELEM --> STACK
(OUT)       POP: STACK --> STACK
(OUT)       TOP: STACK --> ELEM
(OUT)       BOTTOM: STACK --> ELEM
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list limited-stack total
(OUT)    spec LIMITED-STACK
(OUT)        /* STANDART ALGORITHMIC DEFINITION OF A LIMITED-STACK. PUSH ON A FULL  */
(OUT)        /* STACK, POP OR TOP OF AN EMPTY STACK RESULT IN ERRORS */
(OUT)       use   ELEM
(OUT)             LIMIT ;
(OUT)       sorts STACK;
(OUT)       ops   EMPTY:  --> STACK
(OUT)             EMPTY?,FULL?: STACK  --> BOOL
(OUT)             PUSH: STACK ELEM  --> STACK
(OUT)             POP: STACK  --> STACK
(OUT)             TOP: STACK  --> ELEM
(OUT)             BOTTOM: STACK  --> ELEM;
(OUT)    spec-body
(OUT)       constructors EMPTY
(OUT)                    PUSH;
(OUT)       auxiliaries DEPTH: STACK  --> NAT;
(OUT)       define-auxiliaries
(OUT)           DEPTH(ST) = case ST is
(OUT)                         * EMPTY : ZERO
(OUT)                         * PUSH(ST0,EL0) : SUC(DEPTH(ST0))
(OUT)                       esac;
(OUT)       define-carriers
(OUT)    000023 RECORDS PRINTED. CONTINUE?
(OUT)           IS-STACK(ST) = case ST is
(OUT)                           * PUSH(ST0,EL0) : if IS-STACK(EL0)
(OUT)                                             then FALSE
(OUT)                                             else (DEPTH(ST0) LT LIMIT)
(OUT)                           otherwise TRUE
(OUT)                         esac;
(OUT)       define-constructor-ops
(OUT)           EMPTY = * EMPTY
(OUT)           PUSH(ST0,EL0) = if (DEPTH(ST0) LT LIMIT)
(OUT)                           then * PUSH(ST0,EL0)
(OUT)                           else ERROR-STACK;
```

```
(OUT)       define-ops
(OUT)           EMPTY?(ST) = case ST is
(OUT)                          * EMPTY : TRUE
(OUT)                          * PUSH(ST0,EL0) : FALSE
(OUT)                        esac
(OUT)           FULL?(ST) = NOT((DEPTH(ST) LT LIMIT))
(OUT)           POP(ST) = case ST is
(OUT)                       * EMPTY : ERROR-STACK
(OUT)                       * PUSH(ST0,EL0) : ST0
(OUT)                     esac
(OUT)           TOP(ST) = case ST is
(OUT)       000045 RECORDS PRINTED. CONTINUE?
(OUT)                       * EMPTY : ERROR-ELEM
(OUT)                       * PUSH(ST0,EL0) : EL0
(OUT)                     esac;
(OUT)       endspec
(OUT)       FILELEVEL:
(OUT)
```

5.4. Sitzungsprotokoll vom Spec-Editlevel

Die Sitzung aus 5.3. wird vom Benutzer SBQ weitergeführt. Er betritt den EDITORLEVEL für Spezifikationen durch das edit spec Kommando. Die Eingabe des ASPIK-Schlüsselwortes ops positioniert den Editor auf die ops-Klausel, worin SBQ durch das delete Kommando die 6. Deklaration löscht. Danach wird die ops-Klausel automatisch gelistet, in der nun die Operation bottom fehlt. Durch den Aufruf des check Kommandos wird die Korrektheit der Spezifikation LIMITED-STACK nachgeprüft.

```
(OUT)    FILELEVEL:
(OUT)
(IN)      edit spec limited-stack
(OUT)            ************************************************************
(OUT)            ***                                                      ***
(OUT)            ***               S P E C  - E D I T O R                 ***
(OUT)            ***              VERSION VOM:  10. 12. 1984              ***
(OUT)            ***                                                      ***
(OUT)            ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)            ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN       ***
(OUT)            ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)            ***                                                      ***
(OUT)            ************************************************************
(OUT)
(OUT)    POSITION: TOP OF SPEC LIMITED-STACK    LIST
(OUT)    spec LIMITED-STACK
(OUT)        /* STANDART ALGORITHMIC DEFINITION OF A LIMITED-STACK. PUSH ON A FULL  */
(OUT)        /* STACK, POP OR TOP OF AN EMPTY STACK RESULT IN ERRORS */
(OUT)       use   ELEM
(OUT)             LIMIT ;
(OUT)       sorts STACK;
(OUT)       ops   EMPTY:  --> STACK
(OUT)             EMPTY?,FULL?: STACK  --> BOOL
(OUT)             PUSH: STACK ELEM  --> STACK
(OUT)             POP: STACK  --> STACK
(OUT)             TOP: STACK  --> ELEM
(OUT)             BOTTOM: STACK  --> ELEM;
(OUT)    spec-body
(OUT)       constructors EMPTY
(OUT)                    PUSH;
(OUT)       auxiliaries DEPTH: STACK  --> NAT;
(OUT)       define-auxiliaries
(OUT)           DEPTH(ST) = case ST is
(OUT)                         * EMPTY : ZERO
(OUT)                         * PUSH(ST0,EL0) : SUC(DEPTH(ST0))
(OUT)                       esac;
(OUT)
(OUT)    SPEC-EDIT-LEVEL:
(OUT)
```

```
(IN)      ops
(OUT)
(OUT)     POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-STACK    LIST
(OUT)        ops   EMPTY:  --> STACK
(OUT)              EMPTY?,FULL?: STACK  --> BOOL
(OUT)              PUSH: STACK ELEM  --> STACK
(OUT)              POP: STACK  --> STACK
(OUT)              TOP: STACK  --> ELEM
(OUT)              BOTTOM: STACK  --> ELEM;
(OUT)
(OUT)     SPEC-EDIT-LEVEL:
(OUT)
(IN)      delete 6
(OUT)
(OUT)     POSITION: TOP OF OPS-CLAUSE OF SPEC LIMITED-STACK    LIST
(OUT)        ops   EMPTY:  --> STACK
(OUT)              EMPTY?,FULL?: STACK  --> BOOL
(OUT)              PUSH: STACK ELEM  --> STACK
(OUT)              POP: STACK  --> STACK
(OUT)              TOP: STACK  --> ELEM;
(OUT)
(OUT)     SPEC-EDIT-LEVEL:
(OUT)
(IN)      check
(OUT)     %  D800 ERASE FILE CHECKFILE-LIMITED-STACK.00
(OUT)     ***************  C H E C K S Y S T E M   FOR  SPECS  STARTED   ************
(OUT)     ***************                VERSION:  25.02.1985            **************
(OUT)                    **********    USE-CLAUSE CHECKED    **********
(OUT)                    **********   SOPU-CLAUSE  CHECKED   **********
(OUT)                    **********   OPPU-CLAUSE  CHECKED   **********
(OUT)                    **********   PROP-CLAUSE  CHECKED   **********
(OUT)                    **********   CTRS-CLAUSE  CHECKED   **********
(OUT)                    ********** OPS_AUX-CLAUSE CHECKED  **********
(OUT)                    ********** DEF_AUX-CLAUSE CHECKED  **********
(OUT)                    ********** DEF_CAR-CLAUSE CHECKED  **********
(OUT)                    ********** DEF_CTRS-CLAUSE  CHECKED **********
(OUT)                    ********** OPS_PRIV-CLAUSE  CHECKED **********
(OUT)                    **********  DEF_OPS-CLAUSE CHECKED  **********
(OUT)     ***************        E N D      C H E C K S Y S T E M          **************
(OUT)     CPU TIME USED : 32744 ms.
(OUT)     %  D800 ERASE FILE CHECKFILE-LIMITED-STACK
(OUT)
(OUT)                      ************************************************
(OUT)                      * YOUR SPECIFICATION LIMITED-STACK IS CORRECT *
(OUT)                      ************************************************
(OUT)     FILELEVEL:
(OUT)
```

5.5. Sitzungsprotokoll vom Spec-Inputlevel

Das folgende Protokoll zeigt, wie das Eingabesystem für Spezifikationen in SPESY durch das input spec Kommando aufgerufen wird. Nach der Aufforderung einen Kommentar einzugeben, wird der Benutzer nach den Elementen der use-, sorts-, ops-, props- und constructors-Klausel gepromptet, wobei die entsprechenden Schlüsselwörter durch SPESY generiert werden.

Die Operation top wird als Konstruktor abgelehnt, da eine Kontextbedingung die Sorte stack als Zielsorte fordert. Ein Tippfehler in dem Wort stck als Argumentsorte einer auxiliary Operation wird aufgrund einer anderen Kontextbedingung entdeckt.

Zur Definition der auxiliary Operation depth beginnt der Benutzer ein case-Schema, und die linken Seiten aller Fälle werden automatisch erzeugt. Durch wiederholtes Eingeben von ? wird der Benutzer schrittweise über die komplette Syntax von operations-Schemata informiert.

Aufgrund von Kontextbedingungen erkennt das System automatisch das Ende der define-, auxiliaries-, define-carriers- und define-constructors-Klauseln und beginnt mit der nächsten Klausel.

Das Schema, welches das charakteristische Prädikat is-stack definiert, wird außer dem else-Zweig und den fehlenden case-Zweigen vom System generiert. Die Schemata, die die constructor-Operationen push und empty definieren, sind außer dem Term error-stack im else-Zweig von push vollkommen vom System erzeugt.

In der define-ops-Klausel wird der Benutzer an der Definition der nichtdeklarierten Operation bottom gehindert.

Nach dem Ende des Eingabedialogs kann ihn der Benutzer später mit dem ersten Element der define-ops-Klausel fortfahren.

```
(OUT)   FILELEVEL:
(OUT)
(IN)     input spec limited-stack
(OUT)           ************************************************************
(OUT)           ***                                                      ***
(OUT)           ***                S P E C  - E D I T O R                ***
(OUT)           ***               VERSION VOM:  10. 12. 1984             ***
(OUT)           ***                                                      ***
(OUT)           ***   BEI AUFTRETENDEN FEHLERN DES EDITORS ODER DES      ***
(OUT)           ***    EINGABESYSTEMS BITTE EIN PROTOKOLL ERSTELLEN      ***
(OUT)           ***       UND IN GEBAEUDE 14/410 VORBEIBRINGEN !!        ***
(OUT)           ***                                                      ***
(OUT)           ************************************************************
(OUT)   ***************    I N P U T L E V E L     FOR      SPECS     **************
(OUT)   ***************              VERSION :  25.02.1985             **************
(OUT)   ENTER COMMENT OR ;
(OUT)
(IN)    /* STANDART ALGORITHMIC DEFINITION OF A LIMITED-STACK. PUSH ON A FULL STACK,
(IN)    POP OR TOP OF AN EMPTY STACK RESULT IN ERRORS */
(OUT)   use
(OUT)   ENTER SPECTERM COMMENT
(OUT)
(IN)    elem, limit
(OUT)   ENTER SPECTERM COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   sorts
(OUT)   ENTER SORTID COMMENT OR ;
(OUT)
(IN)    stack ;
(OUT)   ops
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    empty: --> stack
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    empty?,full? : stack --> bool
(OUT)   ENTER OP-HEADER COMMENT OR ;
(OUT)
(IN)    push: stack elem --> stack pop: stack --> stack top: stack --> elem ;
(OUT)   props
(OUT)   ENTER PROPERTY COMMENT OR ;
(OUT)
(IN)    ;
(OUT)   spec-body
(OUT)   constructors
(OUT)   ENTER CONSTRUCTOR COMMENT
(OUT)
(IN)    empty push top;
(OUT)
(OUT)   *** INPUT IS IGNORED AT :     ;
(OUT)   *** THE TARGET-SORT OF THE FOLLOWING CONSTRUCTOR IS NOT A NEW SORT ***
(OUT)    TOP
```

```
(OUT)    ENTER CONSTRUCTOR COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    auxiliaries
(OUT)    ENTER AUXILIARY COMMENT OR ;
(OUT)
(IN)     depth:stck --> nat
(OUT)    *** THE FOLLOWING SORTIDS ARE NEITHER IN THE INTERFACE NOR IN SORTS-CLAUSE ***
(OUT)     STCK
(OUT)    ENTER AUXILIARY COMMENT OR ;
(OUT)
(IN)     depth:stack --> nat;
(OUT)    define-auxiliaries
(OUT)    ENTER AUXILIARY-DEFINITION COMMENT
(OUT)
(IN)     depth(st) = case st is
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    OPSCHEME ::= LETSCHEME | CASESCHEME | IFSCHEME | TERM
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    LETSCHEME ::= LET VARID = TERM  , VARID = TERM  ... IN PP OPSCHEME
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    CASESCHEME :: = CASE VARID IS
(OUT)                        PP * LEFTPART : OPSCHEME
(OUT)                         , PP * LEFTPART : OPSCHEME  ...
(OUT)                         , OTHERWISE : OPSCHEME
(OUT)                    PP ESAC
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    IFSCHEME ::= IF TERM
(OUT)                 PP THEN OPSCHEME
(OUT)                  ELSEIF TERM PP THEN OPSCHEME  ...
(OUT)                 PP ELSE OPSCHEME
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    TERM        ::= (TERM) | VARID | PREFIX-TERM | MIXFIX-TERM
(OUT)    PREFIX-TERM ::= OPID-PART   (TERM  , TERM  ... )
(OUT)    MIXFIX-TERM ::=    TERM  , TERM   ... B   OPID-PART B   ...
(OUT)                      TERM  , TERM   ... B   OPID-PART
(OUT)                      B TERM  , TERM   ...
(OUT)    * EMPTY :
(OUT)
(IN)     ?
(OUT)    *** NO MORE DETAILED INFORMATION AVAILAIBLE ***
(OUT)    * EMPTY :
(OUT)
(IN)     zero
```

```
(OUT)    * PUSH ( ST0 , EL0 ) :
(OUT)
(IN)     suc(depth(st0))
(OUT)    ESAC
(OUT)    define-carriers
(OUT)    ENTER CARRIER-DEFINITION COMMENT OR ;
(OUT)
(IN)     is-stack(st) =
(OUT)    CASE ST IS
(OUT)    * PUSH ( ST0 , EL0 ) :
(OUT)    IF ( NOT (IS-STACK (ST0)) )
(OUT)    THEN FALSE
(OUT)    ELSE
(OUT)
(IN)     (depth(st0) lt limit)
(OUT)    ENTER THE MISSING CASE-PARTS OR 'OTHERWISE'
(OUT)
(IN)     otherwise : true
(OUT)    ESAC
(OUT)    define-constructor-ops
(OUT)    ENTER CONSTRUCTOR-DEFINITION COMMENT
(OUT)
(IN)     empty =
(OUT)    * EMPTY
(OUT)    ENTER CONSTRUCTOR-DEFINITION COMMENT
(OUT)
(IN)     push(st0,el0) =
(OUT)    IF DEPTH ( ST0 ) LT LIMIT
(OUT)    THEN * PUSH ( ST0 , EL0 )
(OUT)    ELSE
(OUT)    ENTER OPSCHEME FOR DEFINING THE CONSTRUCTOR ' PUSH '
(OUT)
(IN)     error-stack
(OUT)    private-ops
(OUT)    ENTER PRIVATE-OPERATION COMMENT OR ;
(OUT)
(IN)     ;
(OUT)    define-ops
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     bottom(st) =
(OUT)
(OUT)    *** INPUT IS IGNORED AT :     ST ) =
(OUT)    *** THE FOLLOWING SYMBOL IS NOT A VALID OPID OR OPID-PART ***
(OUT)    BOTTOM
(OUT)    ENTER OP-DEFINITION COMMENT
(OUT)
(IN)     end
(OUT)    ***************          E N D          I N P U T          **************
(OUT)    CPU TIME USED : 34160 ms.
(OUT)
```

5.6. Sitzungsprotokoll vom Map-Input- und Map-Editlevel

Durch das find Kommando erfährt der Benutzer VS, als was und worin die Zeichenfolge elem existiert. Er aktiviert anschließend vom Filelevel aus mit dem input map Kommando das Eingabesystem für Maps. Er gibt die Map elem -is-> nat ein, die nicht OK ist, weil eine Kontextbedingung fordert, daß die benutzte Spezifikation BOOL in der use- oder base-Kausel der Map stehen muß. Die Eingabe eines Fragezeichens veranlaßt das Eingabesystem die an dieser Stelle erwartete ASPIK-Syntax auszugeben. Mit dem Editor für Maps wird sie dann korrigiert, und durch den Aufruf des check Kommandos wird die syntaktische Richtigkeit nachgewiesen. Um die noch fehlende semantische Korrektheit nachzuweisen, wird der MKRP-Beweiser aufgerufen. Das System kehrt automatisch auf den Systemlevel zurück. Die Datei muß dann wieder geöffnet werden. Im vorliegenden einfachen Fall stellt SPESY fest, daß ELEM keine Properties besitzt, und kann deshalb ohne einen Aufruf des MKRP einen Korrektheitsnachweis führen.

```
(OUT)    FILELEVEL:
(OUT)
(IN)      find elem
(OUT)    ELEM IS THE NAME OF A SPEC
(OUT)    ELEM EXISTS IN SPEC ELEM AND IS A SORTNAME.
(OUT)    FILELEVEL:
(OUT)
(IN)      list elem total
(OUT)    spec ELEM
(OUT)        /* LOOSE SPECIFICATION OF JUST A SET */
(OUT)      use   BOOL ;
(OUT)      sorts ELEM;
(OUT)    endspec
(OUT)    FILELEVEL:
(OUT)
(IN)      input map elem -is-> nat
(OUT)     INPUT FOR MAPS AT CMT-CLAUSE STARTED :
(OUT)     CMT
(OUT)
(IN)     /* Schnittstellentest */
(OUT)     IS
(OUT)
```

```
(IN)     refinement
(OUT)     BASE
(OUT)
(IN)
(OUT)
(IN)      ?
(OUT)     SPECTERM |= COMMENT =| ,  ...
(OUT)     OR END BY  ;
(OUT)
(IN)     ;
(OUT)     USE
(OUT)
(IN)     ;
(OUT)    ALL USED SPECTERMS FROM SOURCE HAVE TO BE IN THE BASE-CLAUSE OR EQUAL TO
(OUT)    THE SOURCE OF AN USE-TERM. THE USE-CLAUSE IS NOT OK
(OUT)     SORTS
(OUT)
(IN)     ?
(OUT)     SORT '='  PREFIX'.' SORT ','
(OUT)     OR END BY ';'
(OUT)
(IN)     elem = nat;
(OUT)     OPS
(OUT)
(IN)     ;
(OUT)      ENDMAP ;
(OUT)     THE MAP IS NOT INTERFACE-OK
(OUT)     END OF MAP-INPUT: 1047 MSEC USED
(OUT)    MAP-EDIT: (ELEM -IS-> NAT):
(OUT)
(OUT)    map (ELEM -IS-> NAT)
(OUT)        /* SCHNITTSTELLENTEST */
(OUT)       is REFINEMENT;
(OUT)       sorts ELEM = NAT;
(OUT)    endmap
(OUT)    MAP-EDIT-LEVEL:
(OUT)
(IN)      base
(OUT)    MAP-EDIT: (ELEM -IS-> NAT) AT BASE :
(OUT)
(OUT)
(OUT)    MAP-EDIT-LEVEL:
(OUT)
(IN)      help insert
(OUT)
(OUT)    THE COMMAND "INSERT" ALIAS "I" :
(OUT)    ARGUMENTS : <NUMBER> <TEXT>
(OUT)    <TEXT> IS INSERTED BEFORE THE I-TH ELEMENT OF THE CURRENT CLAUSE -
(OUT)    IF NUMBER IS TO BIG, <TEXT> IS INSERTED BEHIND THE LAST ELEMENT.
(OUT)    MAP-EDIT: (ELEM -IS-> NAT) AT BASE :
(OUT)
(OUT)
```

```
(OUT)    MAP-EDIT-LEVEL:
(OUT)
(IN)      i 1 bool
(OUT)    MAP-EDIT: (ELEM -IS-> NAT) AT BASE :
(OUT)
(OUT)       base
(OUT)           BOOL ;
(OUT)    MAP-EDIT-LEVEL:
(OUT)
(IN)      check
(OUT)      CHECK OF THE MAPID-CLAUSE STARTED ...
(OUT)      CHECK OF THE IS-CLAUSE STARTED ...
(OUT)      CHECK OF THE BASE-CLAUSE STARTED ...
(OUT)      CHECK OF THE USE-CLAUSE STARTED ...
(OUT)      CHECK OF THE SORTS-CLAUSE STARTED ...
(OUT)      CHECK OF THE OPS-CLAUSE STARTED ...
(OUT)    THE MAP IS OK
(OUT)    END OF MAP-EDITOR: 2247 MSEC USED.
(OUT)    FILELEVEL:
(OUT)
(IN)      dir @ *
(OUT)    SPECS ON TESTFILE1          :   LIM-QUEUE
(OUT)    MAPS ON TESTFILE1           :   ELEM.NAT_IS
(OUT)
(OUT)    FILELEVEL:
(OUT)
(IN)      list elem.nat_is
(OUT)    ENTER INFORMATIONPARAMETER :
(OUT)
(IN)      ok
(OUT)             status of map:   (ELEM -IS-> NAT)
(OUT)
(OUT)                                  syntax: +                    semantics
(OUT)                                     |                            |
(OUT)                                     |                            |— consistent: ?
(OUT)                                  interface: +                    |— derivable : ?
(OUT)                                     |
(OUT)                                     |
(OUT)                  |------------------|------------------|
(OUT)                  |                  |                  |
(OUT)               mapid: +           base: +           sorts: +
(OUT)                                  use : 0+          ops  : 0+
(OUT)
(OUT)                    0 = empty
(OUT)                    + = ok
(OUT)                    - = not ok
(OUT)                    ? = unknown
(OUT)
(OUT)
```

```
(OUT)    FILELEVEL:
(OUT)
(IN)      prove elem.nat_is mkrp
(OUT)    %  D800 ERASE FILE LISP.DATA.VS.TESTFILE1.00
(OUT)    SYSTEMLEVEL:
(OUT)
(IN)      open testfile1;list elem.nat_is ok
(OUT)    FILE OPENED !
(OUT)    MESSAGE(S) FOR TESTFILE1 :   NONE
(OUT)
(OUT)              status of map:   (ELEM -IS-> NAT)
(OUT)
(OUT)                                    syntax: +                semantics
(OUT)                                       |                        |
(OUT)                                       |                        |— consistent: +.SPESY
(OUT)                                    interface: +                |— derivable : +.SPESY
(OUT)                                       |
(OUT)                                       |
(OUT)                  |————————————————————|————————————————————|
(OUT)                  |                    |                    |
(OUT)                mapid: +             base: +              sorts: +
(OUT)                                     use : 0+             ops  : 0+
(OUT)
(OUT)                     0 = empty
(OUT)                     + = ok
(OUT)                     - = not ok
(OUT)                     ? = unknown
(OUT)
(OUT)    000021 RECORDS PRINTED. CONTINUE?
(OUT)
(OUT)    FILELEVEL:
```

Literatur

- [BEORSW 85] Breiling, M., Eckl, G., Olthoff, W., Rainau, U., Schmitt, M., Weis, P. : The realisation level. SEKI-Projekt, FB Informatik, Universität Kaiserslautern, 1985.

- [BES 81] Bläsius, K., Eisinger, N., Siekmann, J., Smolka, G., Herold, A., Walther, C. : The Markgraf Carl Refutation Procedure, Proc., 7th IJCAI, 1981

- [BGGORV 83] Beierle,C., Gerlach, M., Göbel, R., Olthoff, W., Raulefs, P., Voß, A. : Integreated Program Development and Verifikation: IN: H. L. Hausen (ed.): Symposium on Software Validation, North-Holland Publ. Co., Amsterdam 1983

- [BGV 83] Beierle, Ch., Gerlach, M., Voß, A. : Parameterization without Parameters-in : The History of a Hierarchy of Specifications. SEKI-Projekt, Memo SEKI-83-09, Universität Kaiserslautern, Fachbereich Informatik, September 1983

- [BV 81/84] Beierle, C., Voß, A. : Entwurfsbeschreibung, Arbeitsunterlagen und Fallstudien zum Spezifikationsentwicklungssystem SPESY, SEKI-Projekt, Univeratiät Kaiserslautern, Fachbereich Informatik, 1981-1984

- [BV 83a] Beierle, Ch., Voß, A. : Canonical Term Functors and Parameterization-by-use for the Specification of Abstract Data Types. SEKI-Projekt, Memo SEKI-83-07, Universität Kaiserslautern, Fachbereich Informatik, May 1983

- [BV 83b] Beierle,Ch., Voß, A., : Parameterization-by-use for hierarchically structured objects. SEKI-Projekt, Memo SEKI-83-08, Universität Kaiserslautern, Fachbereich Informatik, May 1983

- [BV 85] Beierle, C., Voß, A. : Algebraic specifications and implementations in an integrated software development and verification system, SEKI-Projekt, Universität Kaiserslautern, Fachbereich Informatik, 1985

- [Ge 83] Gerlach, M. : A Second-Order Matching Procedure for the Practical Use in a Program Transformation System. SEKI-Projekt, Memo SEKI-83-13, Universität Kaiserslautern, Fachbereich Informatik, September 1983

- [Gei 84] Geisler, C. : MADRE - Ein Programmtransformationssystem für ASPIK-Spezifikationen, SEKI-Projekt, Universität Kaiserslautern, FB Informatik, 1984.

- [Int] SIEMENS-INTERLISP, Benutzerhandbuch Version 4.0, August 1980

- [KRST 83] Kücke, R., Rome, E., Sommer, W., Thomas, Ch. : Das SPEC-System (SPESY) : Benutzerhandbuch, SEKI-Projekt, Universität Kaiserslautern, Fachbereich Informatik, 1983

- [Lic 85] Lichter, H. : Ein interaktives und syntaxorientiertes Eingabesystem für algebraische und algorithmische Spezifikationen, SEKI-Projekt, Universität Kaiserslautern, FB Informatik, 1985.

- [Pet 83] Petersen, U. : Elimination von Rekursionen, SEKI-Projekt, Memo SEKI-83-10, Universität Kaiserslautern, FB Informatik, 1983.

- [Schö 85] Schölles, V. : Beschreibung des Spezifikationsentwicklungssystems SPESY und seiner Implementierung, SEKI-Projekt, Universität Kaiserslautern, FB Informatik, 1985.

- [Som 84] Sommer, W. : SPESY - ein interaktives System zur Unterstützung integrierter Programmspezifikation und Programmverifikation, SEKI-Projekt, Memo SEKI-84-02, Universität Kaiserslautern, Fachbereich Informatik, 1984

- [Spa 85] Spang, H. : Implementierung einer Komponente für die Verwaltung und Modifizierung der Semantik algebraischer Verfeinerungsrelationen und deren Korrektheitsüberprüfungen im Spezifikationssystem SPESY, Projektarbeit, Universität Kaiserslautern, FB Informatik, 1985.

- [SPESY 85] The SPESY System group : SPESY - Systembeschreibung und Dokumentation, SEKI-Projekt, Universität Kaiserslautern, FB Informatik, 1985.

- [Tho 84] Thomas, Ch. : RRLab- Rewrite Rule Labor. Entwurf, Spezifikation und Implementierung eines Softwarewerkzeuges zur Erzeugung und Vervollständigung von Rewrite-Rule Systemen. SEKI-Projekt, Memo SEKI-84-01, Universität Kaiserslautern, Fachbereich Informatik, 1984