

## 3 Grundlagen der ABAP-Programmierung

*Seit Release 4.5 ist ABAP eine hybride Programmiersprache, in der sowohl prozedural als auch objektorientiert programmiert werden kann. Die Laufzeitumgebung, in der ein ABAP-Programm abläuft, ist jedoch in beiden Fällen dieselbe. Wie diese Laufzeitumgebung funktioniert, wie ABAP-Programme aufgebaut sind und was genau passiert, wenn ein ABAP-Programm abläuft, führt uns letztlich dazu, das prozedurale und objektorientierte Programmiermodell verstehen zu können. Folgen Sie uns also zu einem Blick hinter die Kulissen der ABAP-Laufzeitumgebung.*

### 3.1 Das Umfeld eines ABAP-Programms

Dieser Abschnitt soll Ihnen einen Überblick über die Hard- und Softwarelandschaft geben, in der Ihre ABAP-Programme leben. Auf den ersten Blick scheinen sie »irgendwie« im SAP-System abzulaufen und Ihnen auf dem Bildschirm ein Feedback zu geben. Doch Sie werden sehen, es steckt wesentlich mehr dahinter. Nach der Lektüre dieses Abschnitts werden Sie wissen, wo Ihre ABAP-Programme ablaufen und welche Komponenten des SAP-Systems daran beteiligt sind. Dabei werden Sie viele neue Begriffe und Konzepte kennen lernen, die Ihnen helfen sollen, ein tieferes Verständnis für den Ablauf eines ABAP-Programms im SAP-System zu bekommen.

#### 3.1.1 Die Architektur eines SAP-Systems

Jedes SAP-System basiert auf einer dreistufigen Client-Server-Architektur mit einer Präsentations-, einer Applikations- und einer Datenbankschicht (siehe Abbildung 3.1). Jede einzelne dieser Schichten wird durch eine eigene Softwarekomponente repräsentiert, deshalb wird in diesem Kontext auch oft vom softwareorientierten Client-Server-Prinzip gesprochen. Bei diesem Architekturansatz kann das Gesamtsystem auf verschiedene Rechner verteilt werden oder auch nur auf einem einzigen Rechner laufen.<sup>1</sup> Sowohl in vertikaler Richtung (Schichten) als auch in horizontaler Richtung (Komponenten) sind viele verschiedene Szenarien möglich. Bei einer in der Praxis häufig vorzufindenden Konfiguration laufen das Datenbank-

---

1. In einer produktiven SAP-Umgebung dürfte das eher selten vorkommen, technisch ist es jedoch absolut möglich.

system und ein Applikationsserver, der spezielle Dienste für die Datenbank zur Verfügung stellt, gemeinsam auf einem Rechner, während alle weiteren Applikationsserver auf eigene Rechner verteilt sind. Die Komponenten der Präsentationsschicht laufen meistens auf den Desktoprechnern der einzelnen Benutzer.

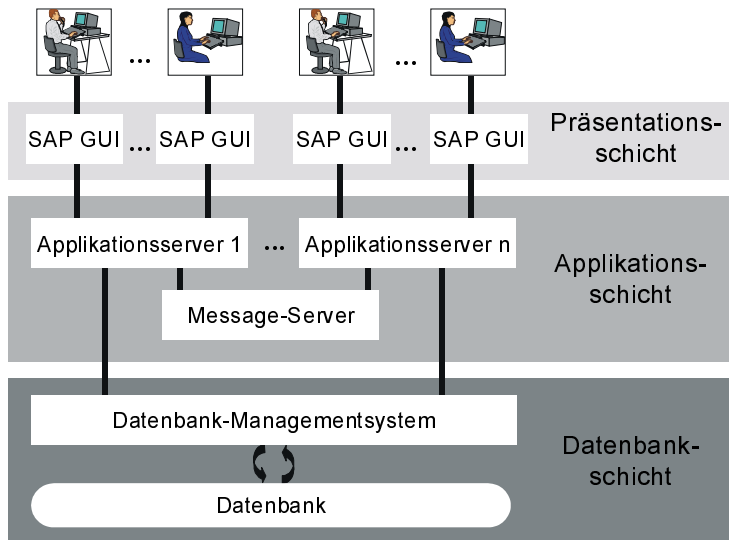


Abbildung 3.1 Die Client-Server-Architektur des SAP-Systems

**Softwareschichten** Jede der Schichten, die in Abbildung 3.1 dargestellt sind, übernimmt eine unterschiedliche Aufgabe im Gesamtsystem, die hier kurz erläutert werden soll.

► **Präsentationsschicht**

Falls Sie gerade vor einem Arbeitsplatz sitzen und dort ein SAP-Bildschirmbild sehen, dann hat die Präsentationsschicht dieses Bild für Sie erzeugt. Die verteilt installierten Softwarekomponenten SAP GUI sorgen dafür, dass die Benutzungsoberfläche des SAP-Systems dargestellt wird und die Tastatureingaben und Mausklicks des Benutzers registriert und an die Applikationsschicht weitergeleitet werden.

► **Applikationsschicht**

Die Applikationsschicht ist der Ort, an dem ein ABAP-Programm seine Arbeit verrichtet. Die Softwarekomponenten der Applikationsschicht bestehen aus einem oder mehreren Applikationsservern und einem Message-Server, der für die Kommunikation zwischen den Applikationsservern zuständig ist. Jeder Applikationsserver stellt eine Reihe von Diensten zum Betrieb des SAP-Systems zur Verfügung. Technisch reali-

siert werden die Dienste eines Applikationsservers durch Workprozesse, deren Anzahl und Typ beim Start des SAP-Systems festgelegt werden. Workprozesse sind Komponenten, die in der Lage sind, eine Anwendung auszuführen. Wichtig ist, dass jeder Workprozess für die gesamte Laufzeit eines SAP-Systems als Benutzer am Datenbanksystem angemeldet ist. Während der Laufzeit des Systems kann eine Datenbankanmeldung nicht von einem Workprozess zum anderen weitergereicht werden.

Im Prinzip genügt ein einziger Applikationsserver, um ein SAP-System zu betreiben. In der Praxis werden die Dienste meistens auf mehrere Applikationsserver verteilt, so dass nicht jeder Applikationsserver jeden Dienst zur Verfügung stellt. Jedes ABAP-Programm wird auf einem Applikationsserver ausgeführt. Es richtet sich nach den Diensten des Applikationsservers, wo ein Programm tatsächlich ausgeführt wird. Ein Programm mit Benutzerdialogen benötigt Dialogdienste, die ein Programm in der Hintergrundverarbeitung nicht braucht.

#### ► **Datenbankschicht**

Jedes SAP-System verfügt über eine zentrale Datenbank, auf welcher der **gesamte** Datenbestand gespeichert ist. Das bedeutet, dass nicht nur die Anwendungsdaten, sondern auch alle Verwaltungsdaten, Customizing-Einstellungen, ABAP-Quelltexte etc. hier enthalten sind. Die Softwarekomponente, die für die Datenbankschicht verantwortlich ist, setzt sich aus dem RDBMS (Relationales Datenbank-Managementsystem) und der eigentlichen Datenbank zusammen.

Für uns als ABAP-Programmierer ist an Abbildung 3.1 am wichtigsten, dass die Applikationsschicht zwischen Präsentations- und Datenbankschicht liegt und dass die einzelnen Applikationsserver über Netzwerkverbindungen mit dem SAP GUI und dem Datenbank-Managementsystem kommunizieren. Dies legt die Rolle von ABAP-Programmen in der Client-Server-Architektur des SAP-Systems eindeutig fest: Die fundamentale Aufgabe von ABAP-Programmen ist die benutzergesteuerte Verarbeitung von Daten aus der Datenbank. Sie erhalten ihre Eingaben aus der Präsentationsschicht und arbeiten mit Daten in der Datenbankschicht. Es gibt natürlich viele weitere Schnittstellen und Erweiterungen zu diesem Bild, auf die wir im Rahmen dieses Buchs aber nicht weiter eingehen wollen.

## 3.2 ABAP-Programme in der Client-Server-Architektur

Die Darstellung der Softwareschichten in Abbildung 3.1 sollte eine Wiederholung allgemein bekannter Tatsachen für Sie sein. Im Folgenden wollen wir näher auf den Einfluss dieser Architektur auf den Aufbau und die Ausführung von ABAP-Programmen eingehen. Wir werden das Bild durch Betrachtung des SAP-Basis-Systems erst verfeinern, um dann durch die Einführung des Begriffs der ABAP-Laufzeitumgebung wieder eine Verallgemeinerung vorzunehmen, die zum Verständnis der ABAP-Programmierung vollständig ausreicht.

### 3.2.1 Das SAP-Basis-System

In diesem Abschnitt wollen wir eine logische Sicht auf die Ausführung von ABAP-Programmen einführen, die nicht auf die technischen Details der Applikationsserver und ihrer Workprozesse eingeht. Hierfür gehen wir zuerst auf das SAP-Basis-System ein. Das SAP-Basis-System ist als Komponente BC Bestandteil jedes SAP-Systems. Es ist die zentrale Plattform aller in ABAP geschriebenen SAP-Anwendungen. In Abbildung 3.2 zeigen wir die logischen Komponenten des SAP-Basis-Systems in der Client-Server-Architektur des SAP-Systems.

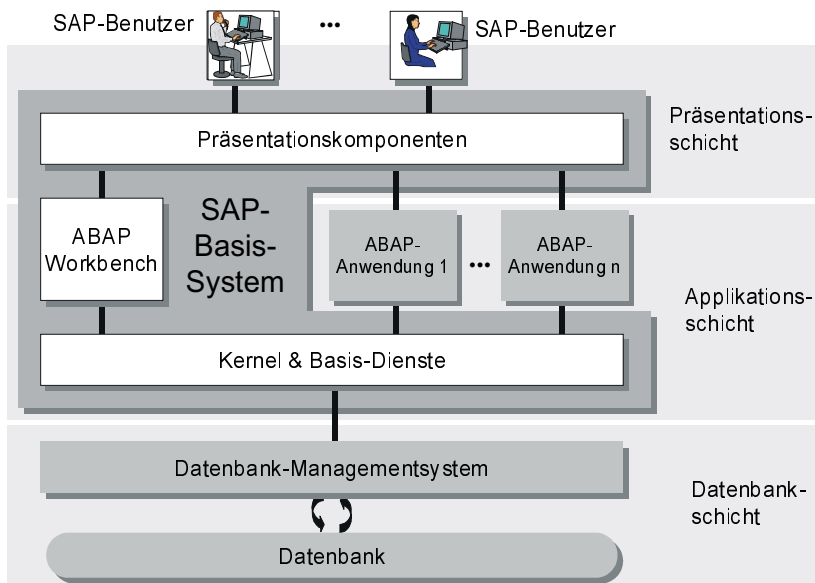


Abbildung 3.2 Das SAP-Basis-System

Das SAP-Basis-System besteht im Wesentlichen aus den folgenden logischen Komponenten:

- ▶ Die **Präsentationskomponenten** dienen der Interaktion von ABAP-Programmen mit dem Benutzer.
- ▶ Die **ABAP Workbench** ist eine komplette Entwicklungsumgebung für ABAP-Programme. Sie ist vollständig in das SAP-Basis-System integriert und wie andere SAP-Anwendungen selbst in ABAP geschrieben.
- ▶ Der **Kernel** dient als hardware-, betriebssystem- und datenbankunabhängige Plattform aller ABAP-Programme. Diese Laufzeitumgebung ist hauptsächlich in C/C++ geschrieben. Alle ABAP-Programme laufen auf Softwareprozessoren (Virtual Machines, VM) dieser Komponente. Die **Basis-Dienste** verwalten Benutzer und Prozesse, stellen eine Datenbankschnittstelle bereit, bieten Kommunikationsschnittstellen mit anderen (SAP-)Systemen und dienen der Administration des SAP-Systems.

Alle in ABAP geschriebenen SAP-Anwendungsprogramme sind in das SAP-Basis-System eingebettet. Dadurch sind sie unabhängig von der Hardware und dem Betriebssystem, aber auch nicht ohne ein SAP-Basis-System ausführbar. Abbildung 3.2 zeigt auch, dass das SAP-Basis-System als Mittler für die Kommunikation zwischen den ABAP-Anwendungen und den Benutzern einerseits und dem Datenbanksystem andererseits dient.

### 3.2.2 Die ABAP-Laufzeitumgebung

Wir können jetzt das Bild aus Abbildung 3.2 wie in Abbildung 3.3 vereinfachen, indem wir sagen, dass jedes ABAP-Programm in eine vom SAP-Basis-System zur Verfügung gestellte Laufzeitumgebung eingebettet ist.

Für ein ABAP-Programm stellt die Laufzeitumgebung die höchste steuernde Instanz dar. Ohne sie könnte kein ABAP-Programm ablaufen. Sie ist für die Ablaufsteuerung von Dialogfolgen verantwortlich, kümmert sich um die Kommunikation mit der Datenbank des SAP-Systems und ist insbesondere für den zeitlichen Ablauf eines ABAP-Programms zuständig.

Abbildung 3.3 ist somit von grundlegender Bedeutung für die gesamte ABAP-Programmierung. Wir haben in dieser Abbildung auch schon den Aufbau von ABAP-Programmen aus Verarbeitungsblöcken angedeutet, auf den wir im Folgenden ausführlich eingehen werden.

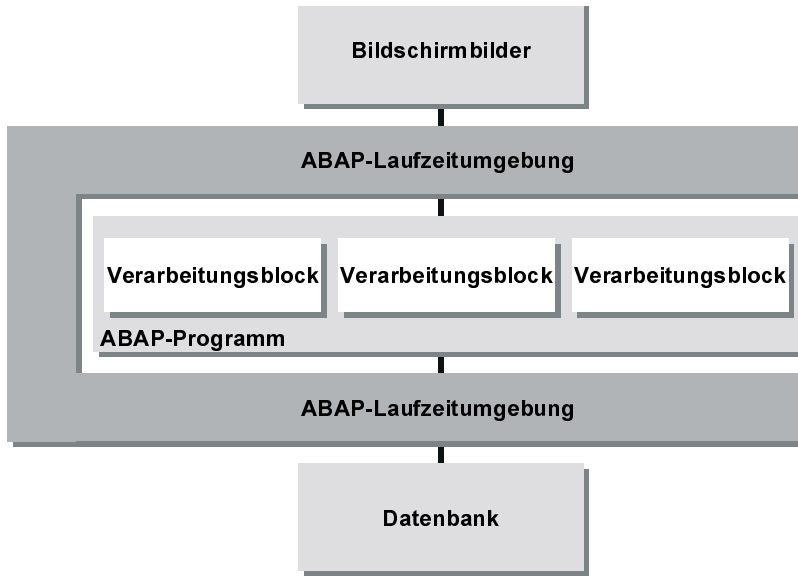


Abbildung 3.3 Die ABAP-Laufzeitumgebung

### 3.3 ABAP-Programme in der Laufzeitumgebung

In diesem Abschnitt gehen wir darauf ein, wie ABAP-Programme aufgebaut sind und wie sie in der Laufzeitumgebung ausgeführt werden. Der Inhalt dieses Abschnitts ist für das Verständnis vorhandener Programme und für das Anlegen neuer Programme unverzichtbar.

#### 3.3.1 Aufbau eines ABAP-Programms

Um zu verstehen, wie ein ABAP-Programm abläuft und welche Rolle die Laufzeitumgebung dabei spielt, gehen wir vom prinzipiellen Aufbau eines ABAP-Programms aus, den wir in Abbildung 3.3 schon angedeutet haben.

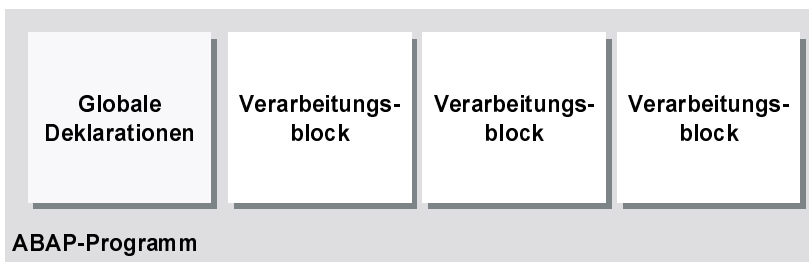


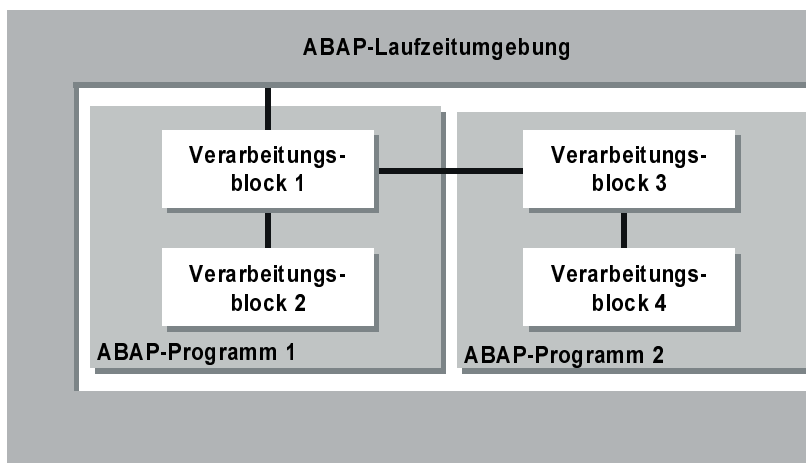
Abbildung 3.4 Der Aufbau eines ABAP-Programms

Jedes ABAP-Programm besteht aus einem globalen Deklarationsteil und einer Reihe von Verarbeitungsblöcken, die je nach Anwendungsfall unterschiedliche Rollen spielen können. Die globalen Objekte sind meistens Datendeklarationen, die in allen Verarbeitungsblöcken des ABAP-Programms sichtbar sind. In bestimmten Verarbeitungsblöcken können lokale Daten deklariert werden, die dann nur lokal, also nur in dem jeweiligen Verarbeitungsblock sichtbar sind.

**Deklarationsteil**

Verarbeitungsblöcke sind unteilbare syntaktische Einheiten. Ein Verarbeitungsblock kann keinen anderen Verarbeitungsblock umfassen. Aufgerufen werden sie entweder durch die Laufzeitumgebung oder durch bestimmte Anweisungen in anderen Verarbeitungsblöcken, Letzteres auch über Programmgrenzen hinweg. Der erste Verarbeitungsblock eines Programms muss also immer durch die Laufzeitumgebung oder aus einem Verarbeitungsblock eines anderen Programms aufgerufen werden. Beim Aufruf des ersten Verarbeitungsblocks eines Programms wird das gesamte Programm in den Speicher geladen.

**Verarbeitungsblock**



**Abbildung 3.5** Aufruf von Verarbeitungsblöcken

Abbildung 3.5 zeigt den Aufruf von Verarbeitungsblöcken. Die Laufzeitumgebung startet Verarbeitungsblock 1 in ABAP-Programm 1. Dieser ruft Verarbeitungsblock 2 im gleichen Programm und Verarbeitungsblock 3 im ABAP-Programm 2. Verarbeitungsblock 3 ruft in seinem Programm Verarbeitungsblock 4 auf. Wenn Sie sich schon etwas mit ABAP auskennen, können Sie Verarbeitungsblock 1 als Ereignisblock START-OF-SELECTION oder als Dialogmodul interpretieren. Verarbeitungsblock 3 ist dann typi-

**Aufruf**

scherweise ein Funktionsbaustein und in ABAP Objects eine Methode einer globalen Klasse. Die Verarbeitungsblöcke 2 und 4 können lokale Unterprogramme oder Methoden lokaler Klassen sein.

- Reihenfolge** Die Reihenfolge, in der die Verarbeitungsblöcke im Programmcode stehen, ist für ihre Ausführungsreihenfolge völlig irrelevant. Nur der Code innerhalb eines Verarbeitungsblocks wird sequenziell ausgeführt. Nach Beendigung eines Verarbeitungsblocks wird die Kontrolle an den jeweiligen Aufrufer zurückgegeben.
- Beendigung** Die Ausführung eines Verarbeitungsblocks kann auf zwei Arten beendet werden. Zum einen ist die Ausführung immer dann zu Ende, wenn die letzte Anweisung des Verarbeitungsblocks ausgeführt wurde. Zum anderen können Sie mit den Anweisungen CHECK und EXIT einen Verarbeitungsblock entweder mit oder ohne Bedingungen programmgesteuert verlassen. Bei jeder Beendigung wird die Kontrolle an den Aufrufer des Verarbeitungsblocks zurückgegeben, das heißt, entweder an die Laufzeitumgebung oder den aufrufenden Verarbeitungsblock. Es ist aber zu beachten, dass CHECK und EXIT innerhalb von Schleifen (siehe Abschnitt 4.4.2) nur auf die aktuelle Schleifenverarbeitung wirken.<sup>2</sup>
- Anweisung** Jede erreichbare Anweisung eines ABAP-Programms, die nicht Teil der globalen Datendeklarationen ist, gehört definitiv zu einem Verarbeitungsblock. Sie können ein ABAP-Programm nur dann lesen und verstehen, wenn Sie in der Lage sind, die verschiedenen Verarbeitungsblöcke zu identifizieren, und wenn Sie wissen, zu welchem Zeitpunkt ihre Ausführung angestoßen wird. Bevor wir näher auf die einzelnen Verarbeitungsblöcke eingehen, betrachten wir einmal das Programm in Listing 3.1.

**Listing 3.1** Welche Verarbeitungsblöcke enthält das ABAP-Programm?

```
REPORT s_processing_blocks.

DATA: wa_spfli TYPE spfli,
      wa_sflight TYPE sflight.

SELECT SINGLE *
FROM   spfli
INTO   wa_spfli
WHERE  carrid = 'LH' AND
       connid = '0400'.
```

---

2. Ab Release 6.10 gibt es eine Anweisung RETURN, die kontextunabhängig immer den Verarbeitungsblock verläßt.

```
PERFORM output USING wa_spfli.
```

```
FORM output USING l_spfli TYPE spfli.
```

```
WRITE: / l_spfli-cityfrom, l_spfli-cityto.
```

```
ENDFORM.
```

```
AT LINE-SELECTION.
```

```
SELECT *
```

```
FROM sflight
```

```
INTO wa_sflight
```

```
WHERE carrid = 'LH' AND
```

```
connid = '0400'.
```

```
WRITE: / wa_sflight-fldate,  
        wa_sflight-seatsmax,  
        wa_sflight-seatsocc.
```

```
ENDSELECT.
```

Können Sie hier schon sämtliche Verarbeitungsblöcke erkennen? Der Block FORM-ENDFORM und die Anweisungen hinter AT LINE-SELECTION sind relativ leicht identifizierbar. Wohin gehören aber die erste SELECT-Anweisung und PERFORM? Setzen Sie einen Breakpoint auf die Anweisung SELECT, starten Sie das Programm und wählen Sie im ABAP Debugger **Übersicht** aus (siehe Abbildung 3.6).

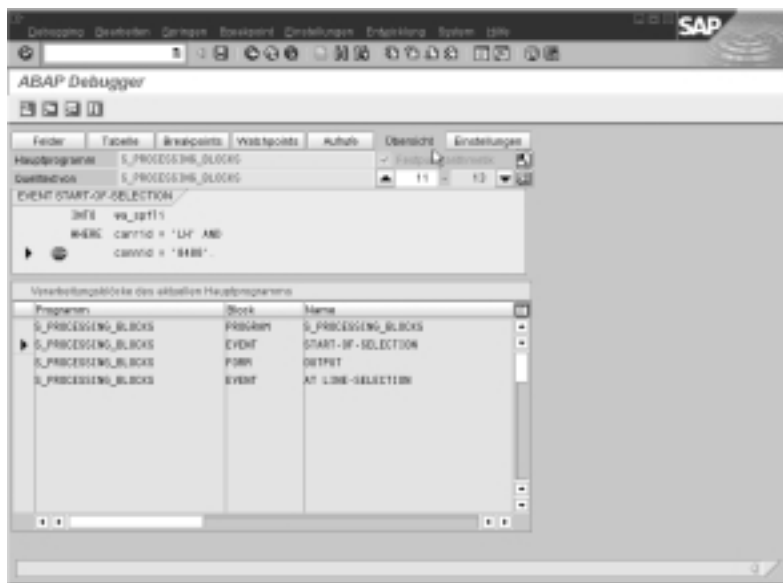


Abbildung 3.6 Verarbeitungsblöcke im Debugger

Im unteren Teil des Bildschirmbilds sehen Sie eine Liste sämtlicher Verarbeitungsblöcke des ABAP-Programms und eine Markierung, in welchem Verarbeitungsblock sich die aktuelle Anweisung befindet. Die Anweisung SELECT gehört also zum Ereignisblock START-OF-SELECTION. Der Verarbeitungsblock START-OF-SELECTION ist der Standardverarbeitungsblock eines ABAP-Programms, dem sämtliche ABAP-Anweisungen zugeordnet werden, die nicht zum globalen Deklarationsteil gehören und vor dem ersten explizit definierten Verarbeitungsblock stehen. Der Anweisungsblock SELECT-ENDESELECT bildet übrigens keinen Verarbeitungsblock, sondern stellt eine ganz normale Schleife dar, die iterativ verarbeitet wird.

**Totes Coding** Beachten Sie aber, dass Anweisungen, die zwischen oder nach abgeschlossenen Verarbeitungsblöcken stehen – in Listing 3.1 wäre das zwischen ENDFORM und AT LINE-SELECTION – nicht erreichbar sind und nie ausgeführt werden. Solches totes Coding wird von der Syntaxprüfung gemeldet.

Die beiden DATA-Anweisungen in Listing 3.1 bilden den globalen Datendeklarationsteil des ABAP-Programms. Ihre Stellung im Programm ist syntaktisch nicht vorgeschrieben. Alle Deklarationsanweisungen eines Programms, die nicht innerhalb von Verarbeitungsblöcken stehen, welche lokale Daten halten können, gehören zum globalen Deklarationsteil, wobei eine Datendeklaration aber immer erst ab der entsprechenden Programmzeile bekannt ist. Wir empfehlen jedoch dringend, sämtliche globalen Datendeklarationen immer zu Beginn des Programms vor dem ersten Verarbeitungsblock aufzuführen.

**Listing 3.2** Kenntlich gemachte Verarbeitungsblöcke

```
*&-----*
*& Report  S_PROCESSING_BLOCKS_COMPLETE      *
*&-----*
```

**REPORT** s\_processing\_blocks\_complete.

\* Global Data Declarations -----

**DATA:** wa\_spfli **TYPE** spfli,  
 wa\_sflight **TYPE** sflight.

\* Standard event -----

**START-OF-SELECTION.**

```

SELECT SINGLE *
FROM   spfli
INTO   wa_spfli
WHERE  carrid = 'LH' AND
       connid = '0400'.

PERFORM output USING wa_spfli.

* Subroutine for List Output -----

FORM output USING l_spfli TYPE spfli.
  WRITE: / l_spfli-cityfrom, l_spfli-cityto.
ENDFORM.

* Reaction on Mouse Doubleclick -----

AT LINE-SELECTION.
  SELECT *
  FROM   sflight
  INTO   wa_sflight
  WHERE  carrid = 'LH' AND
        connid = '0400'.
        WRITE: / wa_sflight-fldate,
                wa_sflight-seatsmax,
                wa_sflight-seatsocc.
  ENDSELECT.

* End of Program -----

```

In Listing 3.2 haben wir das Programm aus Listing 3.1 vervollständigt, indem wir die fehlende Anweisung `START-OF-SELECTION` eingefügt haben. Zudem haben wir die Abgrenzungen der einzelnen Verarbeitungsblöcke durch Kommentarzeilen gekennzeichnet. Das Programm hat genau die gleiche Funktionalität und behält diese auch dann, wenn Sie die Reihenfolge der Verarbeitungsblöcke im Programm ändern, also beispielsweise den Block `AT LINE-SELECTION` vor `START-OF-SELECTION` oder vor die Anweisung `FORM` kopieren. Probieren Sie es aus!

Wenn Sie neue Programme anlegen, sollten Sie die syntaktischen Freiheiten von ABAP nicht ausnutzen, sondern alle Verarbeitungsblöcke durch Kommentare kennzeichnen und auf Standardmechanismen wie das Auf-

**Kennzeichnung**

sammeln nicht zugeordneter Anweisungen verzichten. Dadurch kann ein Leser den Quelltext des Programms viel leichter mit dem Aufbau aus Abbildung 3.4 in Einklang bringen.

### 3.3.2 Verarbeitungsblöcke

Im Folgenden stellen wir Ihnen alle Verarbeitungsblöcke vor, die in ABAP-Programmen möglich sind. Dabei gehen wir kurz darauf ein, welche Anweisungen die Verarbeitungsblöcke definieren, wie sie aufgerufen werden, ob sie lokale Daten halten können und in welchen ABAP-Programmen sie definiert werden können. Wie Sie bald sehen werden, gibt es unterschiedliche Typen von ABAP-Programmen. Der wesentliche Unterschied zwischen den Programmtypen besteht in der Art und Weise, wie die Verarbeitungsblöcke der Programme von der Laufzeitumgebung aufgerufen werden und welche Verarbeitungsblöcke in einem Programm möglich sind.

Wir unterscheiden drei Arten von Verarbeitungsblöcken, die sich durch spezifische Charakteristika voneinander unterscheiden, nämlich **Ereignisblöcke**, **Dialogmodule** und **Prozeduren**. Die genaue Verwendung der Verarbeitungsblöcke wird in den späteren Kapiteln im passenden thematischen Zusammenhang noch genauer erklärt.

#### Ereignisblöcke

Ereignisblöcke werden durch so genannte Ereignisschlüsselwörter eingeleitet. Sie werden nicht explizit durch ein eigenes Schlüsselwort, sondern implizit durch den nächsten Verarbeitungsblock oder das Programmende abgeschlossen. Sie sollten das Ende eines Ereignisblocks im Programm immer durch eine Kommentarzeile kennzeichnen. Ereignisblöcke haben keinen lokalen Datenbereich. Eine Datendeklaration in einem Ereignisblock wird den globalen Daten zugeschlagen!

**Laufzeitereignisse** Die Ausführung eines Ereignisblocks wird durch Ereignisse in der ABAP-Laufzeitumgebung getriggert. Ist zu einem Ereignis der Laufzeitumgebung ein passender Ereignisblock im ABAP-Programm implementiert, wird dieser ausgeführt. Ansonsten hat das Ereignis keine Wirkung. Umgekehrt werden Ereignisblöcke in einem ABAP-Programm, deren Ereignis während der Programmausführung nicht auftritt, auch nicht ausgeführt. Ein typisches Beispiel ist AT LINE-SELECTION in dem obigen Beispielprogramm. Solange der Benutzer auf der Listenausgabe nicht doppelklickt, wird dieser Ereignisblock nicht ausgeführt.

Im Folgenden stellen wir Ihnen die verschiedenen Arten von Ereignisblöcken vor.

► **Programmkonstruktor-Ereignis**

Ereignisschlüsselwort:

**LOAD-OF-PROGRAM.**

Das zugehörige Ereignis tritt während der Programmausführung eines beliebigen Programms außer Classpools genau einmal auf, und zwar dann, wenn das ABAP-Programm in den Speicher geladen wird. Mit den Anweisungen innerhalb dieses Verarbeitungsblocks können Sie die Daten eines Programms initialisieren. Der Programmkonstruktor ist daher mit dem Konstruktor einer Klasse in der objektorientierten Programmierung vergleichbar.

► **Reporting-Ereignisse**

Ereignisschlüsselworte:

**INITIALIZATION.**

**START-OF-SELECTION.**

**GET table.**

**GET table LATE.**

**END-OF-SELECTION.**

Die zugehörigen Ereignisse treten nur bei der Ausführung ausführbarer Programme auf. Das Ereignis **START-OF-SELECTION** nimmt dabei die Sonderrolle des Standardereignisses ein. Das bedeutet, dass alle Anweisungen, die nicht explizit einem anderen Ereignisblock zugeordnet wurden, automatisch an den Ereignisblock **START-OF-SELECTION** angehängt und mit diesem ausgeführt werden. Alle übrigen Reporting-Ereignisse sind für die Implementierung spezieller Anwendungslogiken vorgesehen, die in ausführbaren Programmen auftreten, wenn diese mit logischen Datenbanken verknüpft sind. Mehr dazu finden Sie in Abschnitt 3.3.5.

► **Selektionsbildereignisse**

Ereignisschlüsselwort:

**AT SELECTION-SCREEN ...**

Die zugehörigen Ereignisse werden von der Laufzeitumgebung während der Selektionsbildverarbeitung in ausführbaren Programmen, Modulpools oder Funktionsgruppen ausgelöst. In den Ereignisblöcken können Selektionsbilder vorbereitet oder Benutzeraktionen ausgewertet werden. Mehr dazu finden Sie in Abschnitt 7.2.5.

► **Listenereignisse**

Ereignisschlüsselworte:

**TOP-OF-PAGE.**

**END-OF-PAGE.**

**AT LINE-SELECTION.**

**AT USER-COMMAND.**

Die zugehörigen Ereignisse werden von der Laufzeitumgebung in ausführbaren Programmen, Modulpools oder Funktionsgruppen ausgelöst, während eine klassische Liste erstellt wird oder wenn der Benutzer eine Aktion auf der dargestellten Liste ausführt. In den Ereignisblöcken kann die Liste gestaltet bzw. auf die Benutzeraktionen reagiert werden. Mehr dazu finden Sie in Abschnitt 7.3.7.

► **Dynproereignisse**

Diese Ereignisblöcke werden nicht im ABAP-Programm, sondern in der Dynproablauflogik implementiert. Die Dynproablauflogik steuert die Verarbeitung von Bildschirmbildern in SAP-Systemen. Die Ereignisschlüsselworte der Dynproablauflogik sind:

**PROCESS BEFORE OUTPUT.**

**PROCESS AFTER INPUT.**

**PROCESS ON HELP-REQUEST.**

**PROCESS ON VALUE-REQUEST.**

Die zugehörigen Ereignisse, die abgekürzt auch PBO, PAI, POH und POV heißen, werden von der Laufzeitumgebung während der Dynproverarbeitung ausgelöst. In den Ereignisblöcken der Dynproablauflogik werden die Dialogmodule des ABAP-Programms aufgerufen, während PBO das Bildschirmbild eines Dynpros vorbereiten und während PAI, POH oder POV die Benutzereingaben auswerten. Mehr dazu finden Sie in Abschnitt 7.1.

**Dialogmodule**

Dialogmodule werden in ausführbaren Programmen, Modulpools oder Funktionsgruppen zwischen

**MODULE.**

und

**ENDMODULE.**

implementiert. Wie Ereignisblöcke haben auch Dialogmodule keinen lokalen Datenbereich und auch keine Parameterschnittstelle. Eine Datendeklaration in einem Dialogmodul wird ebenfalls den globalen Daten zugeschlagen!

Dialogmodule werden während obiger Dynproereignisse mit der Anweisung `MODULE` der Dynproablauflogik aufgerufen. Sie enthalten die ABAP-Anweisungen der Dynproverarbeitung. Mehr dazu finden Sie unter 7.1.5.

## **Prozeduren**

Prozeduren sind die Art von Verarbeitungsblöcken, die Sie am ehesten von anderen Programmiersprachen her kennen. Prozeduren haben einen lokalen Datenbereich und können mit einer Parameterschnittstelle versehen werden. Sie können innerhalb desselben Programms oder aus einem anderen ABAP-Programm aufgerufen werden (vergleiche Abbildung 3.5).

Prozeduren stellen eine Möglichkeit dar, ABAP-Programme gezielt zu modularisieren und wieder verwendbare Softwarebausteine bereitzustellen. Das in diesem Abschnitt Gesagte ist natürlich bei weitem nicht alles, was wir Ihnen zum Thema Modularisierungstechniken und Wiederverwendbarkeit von Softwarekomponenten verraten wollen. Für die Beschreibung der folgenden drei Prozedurarten haben wir Kapitel 5 reserviert, wobei wir den Fokus auf die Neuerungen legen werden, die uns durch die Methoden in ABAP Objects zur Verfügung stehen. Zum jetzigen Zeitpunkt interessieren uns Prozeduren nur in ihrer Rolle als Verarbeitungsblöcke.

Lokale Daten

### **► Unterprogramme**

Unterprogramme können in beliebigen ABAP-Programmen außer Classpools zwischen den Anweisungen

`FORM ...`

und

`ENDFORM.`

implementiert werden. Sie werden vom Verwender durch die Anweisung `PERFORM` aufgerufen, wobei die Parameterschnittstelle über positionale Parameter, wie es von C oder Pascal her bekannt ist, angesprochen wird. Mehr dazu finden Sie in Abschnitt 5.2.1.

### ► Funktionsbausteine

Funktionsbausteine können nur in ABAP-Programmen vom Typ Functionpool zwischen den Anweisungen

```
FUNCTION ...
```

und

```
ENDFUNCTION.
```

implementiert werden. Sie werden mit der Anweisung `CALL FUNCTION` aufgerufen, wobei der Verwender die Schnittstelle mit Schlüsselwortparametern versorgt und daher deren Namen kennen muss. Heute ist noch ein großer Teil der gesamten von SAP ausgelieferten Funktionalität in Funktionsbausteinen gekapselt. Mehr dazu finden Sie in Abschnitt 5.2.1.

### ► Methoden

Methoden gehören seit der Einführung von ABAP Objects zum Repertoire der Prozeduren. Sie werden zwischen den Anweisungen

```
METHOD ...
```

und

```
ENDMETHOD.
```

im Implementierungsteil einer Klasse implementiert und mit der Anweisung `CALL METHOD` aufgerufen, wobei der Verwender die Schnittstelle mit Schlüsselwortparametern versorgt. Funktionale Methoden, die nur einen speziellen Ausgabeparameter haben, können auch als Operanden in Ausdrücken verwendet werden. Wenn Methoden als Ereignisbehandler deklariert sind, können sie auch durch Ereignisse in ABAP Objects getriggert werden. Schließlich können wir noch Instanzmethoden und statische Methoden unterscheiden, die entweder an Objekte oder nur an die Klasse gebunden sind. Mehr dazu finden Sie in Abschnitt 5.3.

## Zur Rolle von Klassen

Seit der Einführung von ABAP Objects können in ABAP-Programmen Klassen definiert werden. Wie verhalten Klassen sich in Bezug auf den Programmaufbau aus Verarbeitungsblöcken? Klassen werden zwischen den Anweisungen `CLASS` und `ENDCLASS` definiert, wobei es für jede Klasse einen Deklarationsteil und einen Implementierungsteil gibt. Klassen sind wie Verarbeitungsblöcke nicht schachtelbar. Wir können die Definition einer Klasse als Klammer um bestimmte Programmkomponenten verstehen, wodurch eine neue syntaktische Einheit erzeugt wird (siehe Abbildung 3.7).

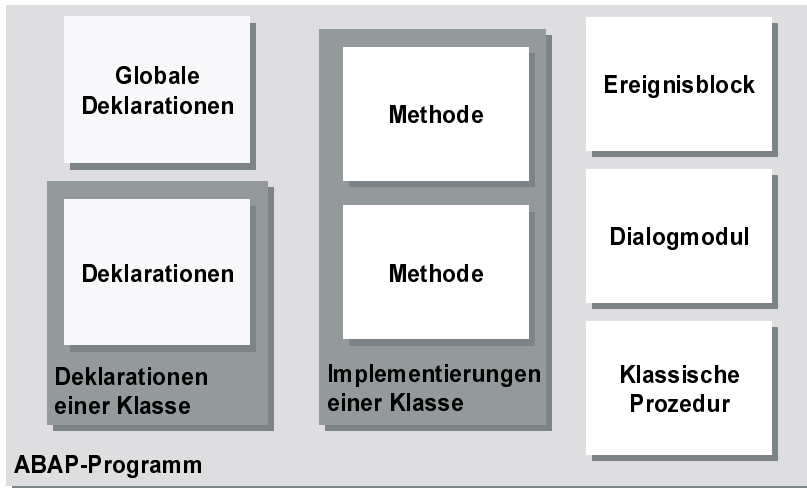


Abbildung 3.7 Klassen im Programmaufbau

Der Deklarationsteil einer Klasse wird mit `CLASS ... DEFINITION` eingeleitet. Er bindet Deklarationen in eine Klasse ein und verändert damit deren Sichtbarkeit. Ein Deklarationsteil mit seinen Datendeklarationen gehört zu den übrigen globalen Deklarationen eines Programms und sollte auch dort angeordnet werden.

Deklaration

Der Implementierungsteil einer Klasse wird mit `CLASS ... IMPLEMENTATION` eingeleitet. Er bindet die Methoden der Klasse ein und gehört syntaktisch somit zu den Verarbeitungsböcken des Programms. Die Anordnung des Implementierungsteils in Bezug auf die anderen Verarbeitungsböcke ist genauso beliebig wie die Reihenfolge der Methoden innerhalb des Implementierungsteils selbst. Der besseren Lesbarkeit wegen empfehlen wir aber, die Implementierung in der Nähe der Deklaration aufzuführen.

Implementierung

Abbildung 3.7 zeigt, dass Klassen vollständig in den klassischen Programmaufbau integriert sind und parallel zu den klassischen Komponenten angelegt werden können.

### 3.3.3 Programmtypen

Jedes ABAP-Programm hat einen Typ, der in den Programmeigenschaften festgelegt wird (siehe Abbildung 3.8). Vor Release 4.6 wurden hier noch die internen einstelligen Kürzel angegeben. Ein »Ausführbares Programm« hatte beispielsweise den Typ 1. Inzwischen wird der Programmtyp über den vollen Namen ausgesucht.

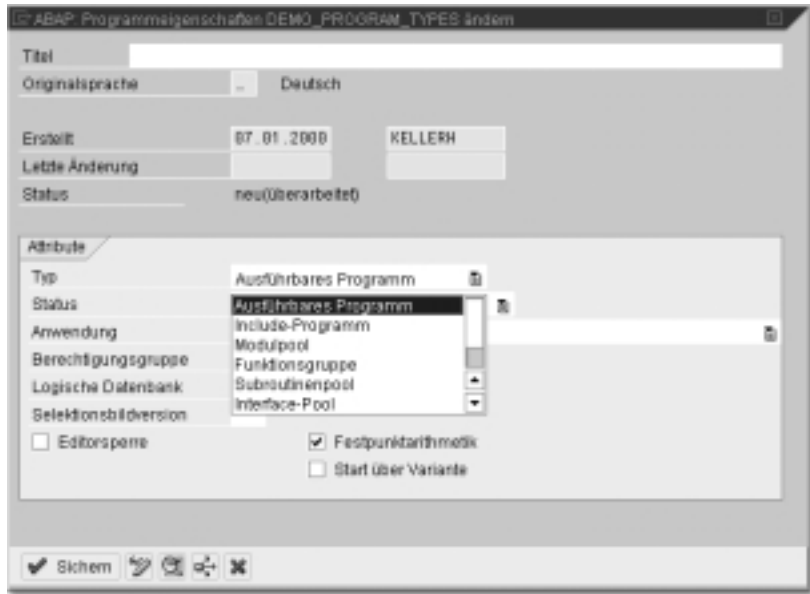


Abbildung 3.8 Programmtyp in den Programmeigenschaften

Was hat es nun mit den verschiedenen Programmtypen auf sich? Der Programmtyp legt fest, welche Verarbeitungsblöcke ein Programm enthalten kann, wie das Programm von der Laufzeitumgebung behandelt bzw. ausgeführt wird und ob es mit eigenen Bildschirmbildern arbeiten kann. Es gibt folgende ABAP-Programmtypen:

### **Ausführbare Programme (Typ 1)**

Ausführbare Programme werden mit der Anweisung REPORT eingeleitet. Sie können eigene Bildschirmbilder enthalten und sind über die Anweisung SUBMIT oder über Transaktionscodes ausführbar. Ausführbare Programme können außer Funktionsbausteinen alle in ABAP möglichen Verarbeitungsblöcke und beliebig viele lokale Klassen enthalten. Sie werden direkt mit dem Werkzeug ABAP Editor angelegt. Während ihrer Ausführung können sämtliche Ereignisse der Laufzeitumgebung auftreten.

### **Modulpools (Typ M)**

Modulpools werden mit der Anweisung PROGRAM eingeleitet. Sie können eigene Bildschirmbilder enthalten und sind nur über Transaktionscodes ausführbar. Modulpools können außer Reporting-Ereignisblöcken und Funktionsbausteinen alle in ABAP möglichen Verarbeitungsblöcke und

beliebig viele lokale Klassen enthalten. Während ihrer Ausführung können sämtliche Ereignisse der Laufzeitumgebung außer Reporting-Ereignissen auftreten. Sie werden direkt mit dem ABAP Editor angelegt.

### **Funktionsgruppen (Typ F)**

Funktionsgruppen bzw. Functionpools werden mit der Anweisung FUNCTION-POOL eingeleitet. Sie können eigene Bildschirmbilder enthalten. Funktionsgruppen werden in der Regel aber nicht direkt ausgeführt, sondern durch den Aufruf ihrer Funktionsbausteine geladen. Sie können außer Reporting-Ereignisblöcken alle in ABAP möglichen Verarbeitungsblöcke und beliebig viele lokale Klassen enthalten. Sie sind die einzigen Programme, die Funktionsbausteine enthalten können, und sie werden mit dem Werkzeug Function Builder angelegt.

### **Classpools (Typ K)**

Classpools werden mit der Anweisung CLASS-POOL eingeleitet. Sie können keine eigenen Bildschirmbilder und keine Verarbeitungsblöcke außer Methoden enthalten. Classpools können genau eine globale Klasse und beliebig viele lokale Klassen enthalten. Sie werden nicht direkt ausgeführt, sondern durch die Verwendung ihrer globalen Klasse geladen.<sup>3</sup> Sie werden mit dem Werkzeug Class Builder angelegt.

### **Interfacepools (Typ J)**

Interfacepools werden mit der Anweisung INTERFACE-POOL eingeleitet. Sie können keine eigenen Bildschirmbilder und keinerlei Verarbeitungsblöcke enthalten. Sie enthalten die Definition genau eines globalen Interfaces, das in beliebigen globalen oder lokalen Klassen implementiert werden kann. Sie werden mit dem Werkzeug Class Builder angelegt.

### **Subroutinenpool (Typ S)**

Subroutinenpools werden mit der Anweisung PROGRAM eingeleitet. Sie können keine eigenen Bildschirmbilder und außer dem Ereignisblock LOAD-OF-PROGRAM nur Unterprogramme als Verarbeitungsblöcke enthalten. Subroutinenpools werden nicht direkt ausgeführt, sondern durch den externen Aufruf ihrer Unterprogramme geladen. Sie werden mit dem ABAP Editor angelegt.

---

3. Ab Basis-Release 6.10 wird es aber auch Transaktionscodes geben, die mit globalen Klassen verknüpft sind und bei ihrer Verwendung implizit ein Objekt der Klasse erzeugen.

## Typgruppen

Typgruppen bzw. Typenpools werden mit der Anweisung TYPE-POOL eingeleitet. Sie können keine eigenen Bildschirmbilder und keinerlei Verarbeitungsblöcke enthalten. Sie enthalten die Definitionen globaler Datentypen, die über die Anweisung TYPE-POOLS in jedem ABAP-Programm sichtbar gemacht werden können. Sie werden mit dem Werkzeug ABAP Dictionary angelegt.

## Include-Programme (Typ I)

Include-Programme haben keine programmeinleitende Anweisung und sind im Gegensatz zu allen anderen Programmtypen keine eigenständigen Kompilationseinheiten mit eigenem Speicherbereich. Include-Programme haben eine reine Bibliotheksfunktion für ABAP-Quelltext und werden über die Anweisung INCLUDE an beliebigen Stellen anderer ABAP-Programme eingebunden. Include-Programme stehen technisch in keinerlei Zusammenhang mit Verarbeitungsblöcken. Es bietet sich aber an, logische Programmeinheiten, wie den Deklarationsteil für globale Daten, mehrere gleichartige oder einzelne Verarbeitungsblöcke in jeweils eigene Include-Programme aufzuteilen. Die ABAP Workbench unterstützt die automatische Aufteilung von Modulpools, Funktionsgruppen und Classpools in Include-Programme. Eigene Include-Programme legen Sie direkt mit dem ABAP Editor an.

### 3.3.4 Bildschirmbilder

Wir haben bisher schon mehrfach die Bildschirmbilder von ABAP-Programmen erwähnt. Da wir der Programmierung und Verarbeitung von Bildschirmbildern einen ganzen Teil dieses Buchs gewidmet haben, wollen hier deshalb nur kurz die möglichen Bildschirmarten aufzählen. Folgende Bildschirmbilder können zu ausführbaren Programmen, Modulpools und Funktionsgruppen gehören und dort verarbeitet werden:

## Dynpros

Dynpros (dynamische Programme) bestehen aus dem eigentlichen Bildschirmbild und einer Ablauflogik. Die Dynproablauflogik ist eine Programmschicht zur Behandlung von Dynproereignissen und liegt zwischen ABAP-Programm und der Laufzeitumgebung. Bildschirmbild und Ablauflogik eines Dynpros werden mit dem Werkzeug Screen Painter angelegt.

## Selektionsbilder

Selektionsbilder sind Spezialdynpros, die mit ABAP-Anweisungen und nicht mit dem Screen Painter angelegt werden. Die Ablauflogik von Selektionsbildern ist in die Laufzeitumgebung eingebunden, wobei die Dynproereignisse in Selektionsbildereignisse verwandelt werden.

## Klassische Listen

Klassische Listen, früher einfach nur Listen genannt, sind Bildschirmbilder eines Spezialdynpros mit einem einzigen Ausgabebereich, der mit ABAP-Anweisungen formatiert beschrieben werden kann. Die Ablauflogik des Dynpros ist wie bei Selektionsbildern in die Laufzeitumgebung eingebunden, und ihre Ereignisse werden als Listenereignisse an das ABAP-Programm weitergegeben.

### 3.3.5 Programmausführung

Aus unserer Beschreibung der Programmtypen geht hervor, dass fast alle Programme zu bereits laufenden Programmen hinzugeladen werden. Nur ausführbare Programme und Modulpools sind aus Benutzersicht ausführbar. Die Ausführung eines Programms durch den Benutzer bedeutet, dass er es vom Bildschirm aus starten kann. In diesem Abschnitt über Programmausführung sind also nur ausführbare Programme und Modulpools von Belang. Die übrigen Programmtypen behandeln wir in Kapitel 5.

Wenn wir uns jetzt wieder auf den Aufbau von ABAP-Programmen aus

**Steuerung**

Verarbeitungsblöcken besinnen und nochmals Abbildung 3.5 betrachten, wird klar, dass die Ausführung eines ABAP-Programms nichts anderes bedeuten kann, als seine Verarbeitungsblöcke in einer bestimmten Reihenfolge aufzurufen. Die Steuerung eines ABAP-Programms erfolgt damit vollständig in der Laufzeitumgebung.

**Prozessor**

Hierfür enthält die Laufzeitumgebung so genannte **Prozessoren**. Es gibt beispielsweise einen Prozessor für die Behandlung von Dynpros oder einen Prozessor für die Verarbeitung von Ausgabelisten. Hier sind die Steuerungsabläufe definiert, die Bildschirme und Verarbeitungsblöcke in bestimmten, zweckgebundenen Reihenfolgen aufrufen können. Die Kontrolle über das Programm liegt immer in der Hand eines bestimmten Prozessors.

Abbildung 3.9 zeigt einige dieser Prozessoren. In diese Abbildung haben wir auch die oben schon erwähnte Dynproablauflogik aufgenommen. Während Selektionsbild- und Listenprozessor auch die Dynproablauflogik Ihrer Bildschirmbilder umfassen und Ereignisse direkt an das ABAP-Pro-

programm senden, müssen Sie die Ablauflogik von Dynpros selbst programmieren. Der Dynproprozessor sendet seine Ereignisse an die Verarbeitungsböcke PBO und PAI, in denen Dialogmodule des ABAP-Programms aufgerufen werden.

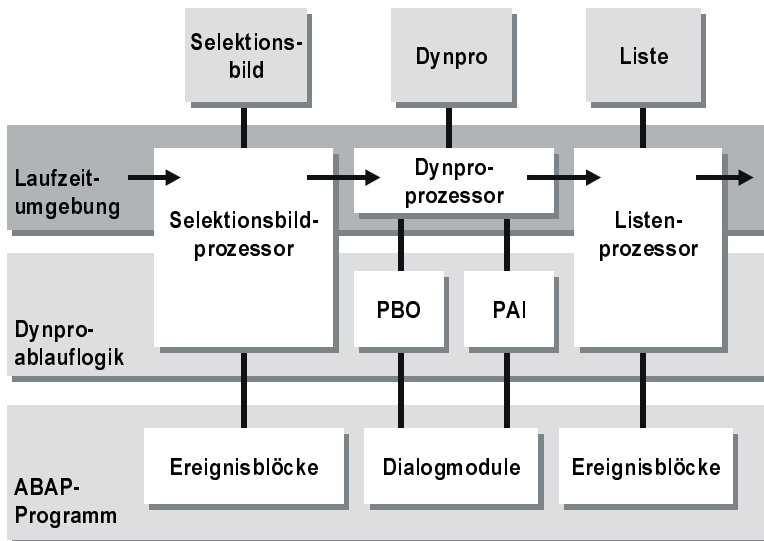


Abbildung 3.9 Prozessoren der Laufzeitumgebung

**Programmstart** Ein ABAP-Programm zu starten bedeutet letztlich nichts anderes, als einen oder mehrere Prozesse hintereinander in der Laufzeitumgebung auszuführen. Welcher Prozess als Erstes gestartet wird, richtet sich primär nach dem Programmtyp und der Art des Programmaufrufs. Die weitere zeitliche Reihenfolge hängt insbesondere bei dialogorientierten Anwendungen auch von den Benutzeraktionen auf den Bildschirmbildern ab.

Wenn wir im allgemeinen Sprachgebrauch von einer ABAP-Programmausführung reden, muss Ihnen nach dem bisher Gesagten jetzt immer klar sein, dass die eigentliche Steuerung durch die Laufzeitumgebung oder durch Bildschirmabläufe stattfindet und dass ABAP-Programme nichts anderes sind als Pools von Verarbeitungsböcken, deren Ausführung bei Bedarf von außen angefordert wird. ABAP-Programme lassen sich zwar auch so programmieren, dass nur der erste Verarbeitungsblock von außen angestoßen wird und alle weiteren Verarbeitungsböcke intern aufgerufen werden. Ein solches Programm bietet dann aber fast keine Schnittstellen zum Benutzer.

Wir gehen jetzt näher darauf ein, wie ausführbare Programme und Modulpools von der Laufzeitumgebung gesteuert werden.

## Der Ablauf ausführbarer Programme

Ausführbare Programme können durch die Eingabe ihres Namens in der Transaktion »SA38« (Pfad: **System** · **Dienste** · **Reporting**) oder über die ABAP-Anweisung **SUBMIT** gestartet werden. Die Transaktion »SA38« macht letztlich auch nichts anderes, als das angegebene Programm über **SUBMIT** aufzurufen. Technisch gesehen ist die Aufrufbarkeit über **SUBMIT** die hauptsächliche Charakteristik eines ausführbaren Programms. Aus Benutzersicht sind ausführbare Programme die einzigen ABAP-Programme, die durch die Eingabe ihres Namens gestartet werden können.

**SUBMIT**

Beim Aufruf eines ausführbaren Programms wird in der Laufzeitumgebung eine fest vordefinierte Folge von Prozessschritten gestartet, die eine vorgegebene Folge von Ereignissen auslösen. Abbildung 3.10 stellt die zeitliche Reihenfolge der Ereignisse in einem Ablaufdiagramm dar. Hier finden Sie viele der in Kapitel 3.3.1 vorgestellten Ereignisse wieder.

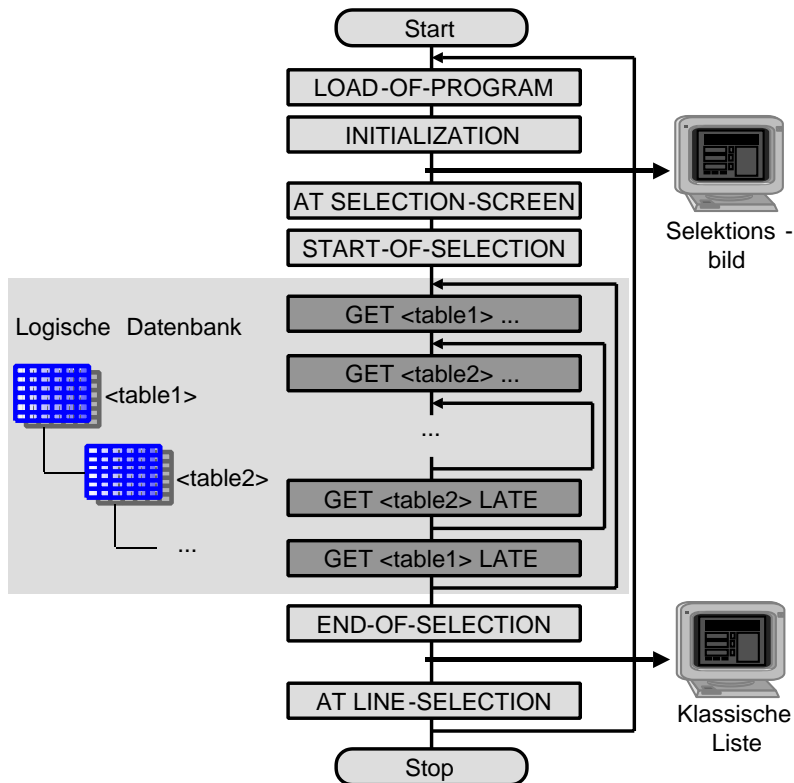


Abbildung 3.10 Ereignisse der Laufzeitumgebung in ausführbaren Programmen

**Selektionsbild** Wie bei allen ABAP-Programmen löst die Laufzeitumgebung als Erstes das Ereignis für den Programmkonstruktor LOAD-OF-PROGRAM aus<sup>4</sup>. Der folgende Ablauf ist aber spezifisch für ausführbare Programme. Falls in dem Programm ein Selektionsbild definiert ist, kann dieses während des Ereignisses INITIALIZATION vorbereitet werden und wird dann von der Laufzeitumgebung automatisch gesendet. Benutzeraktionen auf dem Selektionsbild führen zu den Ereignissen AT SELECTION-SCREEN. Nach Beendigung der Selektionsbildverarbeitung wird das Standardereignis START-OF-SELECTION ausgelöst.

**Logische Datenbank** Die folgenden Ereignisse werden dann und nur dann ausgelöst, wenn das ausführbare Programm in seinen Eigenschaften mit einer logischen Datenbank verknüpft ist. Eine logische Datenbank ist ein spezielles ABAP-Programm, das die Definition eines Selektionsbilds und eine Reihe von Unterprogrammen enthält. Die Unterprogramme enthalten ABAP-Anweisungen, die in der Regel dem Auslesen von Daten aus Datenbanktabellen dienen. Bei der Ausführung eines ausführbaren Programms mit einer logischen Datenbank wird das Selektionsbild der logischen Datenbank gesendet. Dann ruft die Laufzeitumgebung die Unterprogramme der logischen Datenbank in einer Reihenfolge auf, die sich aus einer in der logischen Datenbank festgelegten hierarchischen Struktur ergibt. Wenn ein Unterprogramm der logischen Datenbank eine Zeile einer Datenbanktabelle `table` eingelesen hat, löst es mit der Anweisung `PUT table` das Ereignis `GET table` der Laufzeitumgebung aus. Dies triggert den entsprechenden Ereignisblock des ausführbaren Programms, wo die eingelesenen Daten in einem gemeinsamen Datenbereich ausgewertet werden können. Eine logische Datenbank ist also nichts anderes als eine wieder verwendbare Komponente, welche Details der Datenselektion kapselt.

**LDB\_PROCESS** Die Methode, nach der die logische Datenbank ihre Daten bereitstellt, ist nicht auf Datenbankzugriffe beschränkt. Möglich sind auch Zugriffe auf beliebige Datenablagen oder sogar die Generierung eigener Daten. Seit Basis-Release 4.0 sind logische Datenbanken auch unabhängig von einem ausführbaren Programm über den Funktionsbaustein LDB\_PROCESS in beliebigen Programmen aufrufbar. Sie pflegen logische Datenbanken in der Transaktion SE36. Wir werden in diesem Buch nicht weiter auf dieses Thema eingehen, sondern wollen Sie an dieser Stelle auf die Dokumentation dieser Transaktion verweisen.

---

4. Seit Release 4.6

Wenn die logische Datenbank ihre Datenselektion beendet hat, löst die Laufzeitumgebung das Ereignis END-OF-SELECTION aus. Im zugehörigen Ereignisblock können die eingelesenen Daten in der Summe bearbeitet werden. Ohne Anbindung an eine logische Datenbank kommt END-OF-SELECTION gleich hinter START-OF-SELECTION. Die Implementierung des zugehörigen Ereignisblocks ist dann nicht notwendig, da das gesamte Coding genauso gut dem Ereignisblock für START-OF-SELECTION angefügt werden kann.

Zum Abschluss eines ausführbaren Programms stellt die Laufzeitumgebung automatisch die während der Ausführung beschriebene klassische Liste dar. Benutzeraktionen auf der Liste führen zu Listenereignissen. Wenn der Benutzer die Liste verlässt und kein Selektionsbild im Programm definiert ist, wird das Programm beendet. Ansonsten startet die Laufzeitumgebung das Programm erneut. Es kann dann nur beendet werden, indem der Benutzer die Bearbeitung des Selektionsbilds abbricht.

Liste

Der vorgegebene Ablauf eines ausführbaren Programms unterstützt mit der Möglichkeit zur Eingabe von Selektionsparametern auf einem Selektionsbild, der anschließenden Datenverarbeitung und der abschließenden Datenausgabe auf einer Liste die klassischen Aufgaben des Reportings. Wir nennen den Prozessor der Laufzeitumgebung, der den Ablauf aus Abbildung 3.10 erzeugt, deshalb auch Reporting-Prozessor.

Mit dem hier beschriebenen Ablauf können Sie jetzt auch unser allererstes Programm S\_VERY\_FIRST\_PROGRAM aus Abschnitt 2.3.3 und damit alle ähnlich aufgebauten Programme verstehen. In diesem Programm haben wir mit PARAMETERS ein Selektionsbild definiert, einen einzigen Verarbeitungsblock START-OF-SELECTION implementiert und in diesem mit WRITE eine Liste beschrieben. Für dieses Programm reduziert sich Abbildung 3.10 auf ein einziges Ereignis, nämlich START-OF-SELECTION, und die beiden Bildschirmbilder. Bei der Ausführung wird erst das Selektionsbild angezeigt, dann START-OF-SELECTION ausgeführt, danach die Liste angezeigt und das Programm schließlich erneut gestartet, bis der Benutzer das Selektionsbild verlässt.

### **Der Ablauf von Modulpools**

Die Ausführung eines Modulpools ist sehr unterschiedlich im Vergleich zu ausführbaren Programmen. Bei der Ausführung eines Modulpools wird in der Laufzeitumgebung kein Prozess gestartet, der mit einem vorgegebenen Zeitablauf Ereignisse an das ABAP-Programm sendet. Modulpools lassen sich auch nicht über die Angabe ihres Namens oder SUBMIT starten.

**Transaktionscode** Um einen Modulpool zu starten, müssen Sie im Modulpool mindestens ein Dynpro anlegen und einen Transaktionscode definieren, der mit einem der Dynpros des Modulpools verknüpft ist. Dieses Dynpro wird dadurch zum Einstiegsdynpro einer Transaktion, und es kann weitere Dynpros des Modulpools aufrufen.

Sie starten eine Transaktion, indem Sie den Transaktionscode in das Eingabefeld der Symbolleiste eingeben oder einen Menüeintrag auswählen, der mit der Transaktion verknüpft ist. Beim Start einer Transaktion lädt die Laufzeitumgebung das Programm in den Speicher (wobei LOAD-OF-PROGRAM ausgelöst wird) und löst das Ereignis PBO des Einstiegsdynpros aus. Nach der Verarbeitung dieses Ereignisses in der Dynproablauflogik wird das Bildschirmbild des Dynpros gesendet. Benutzeraktionen auf dem Bildschirm lösen in der Laufzeitumgebung das Ereignis PAI aus. Nach dessen Verarbeitung löst die Laufzeitumgebung das Ereignis PBO des nächsten Dynpros aus, und der Vorgang wiederholt sich, bis das Programm durch eine Anweisung wie LEAVE PROGRAM oder dadurch, dass das letzte Dynpro erreicht wurde, beendet wird.

**Modulpool** Bei dieser Art von Programmausführung beschränkt sich die Rolle des ABAP-Programms auf die Bereitstellung der Dialogmodule, die in der Dynproablauflogik aufgerufen werden. Daher der Name Modulpool. Die Aufrufreihenfolge wird durch aufeinander folgende Dynpros geprägt, deren Reihenfolge in der Regel stark von den Aktionen des Benutzers abhängig ist.

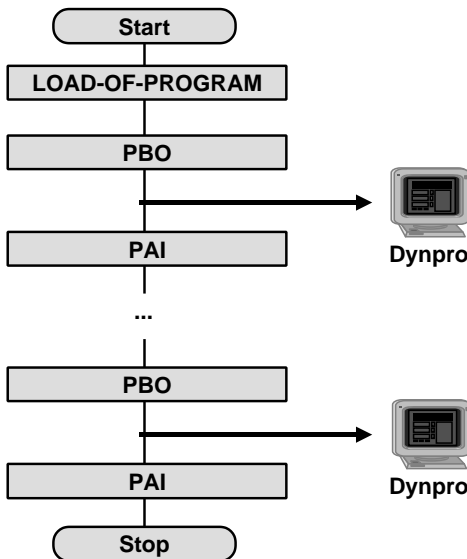


Abbildung 3.11 Ereignisse der Laufzeitumgebung in Modulpools

So lange keine Selektionsbilder mit der Anweisung CALL SELECTION-SCREEN oder Listen mit der Anweisung LEAVE TO LIST-PROCESSING aufgerufen werden, wird die Ausführung eines Modulpools nur vom Dynprozessor der Abbildung 3.9 gesteuert. In der zeitlichen Reihenfolge der Ereignisse wirkt sich das so aus, dass sich immer PBO und PAI aufeinander folgender Dynpros abwechseln (siehe Abbildung 3.11).

## Vergleich der Programmausführungen

Während die Ereignissteuerung eines ausführbaren Programms vordergründig einer vorgegebenen Folge von Prozessschritten unterliegt, hat der Entwickler bei Modulpools alle Freiheiten, durch entsprechende Programmierung der Dynproablauflogik und der zugehörigen Dialogmodule den Programmablauf selbst zu gestalten.

Nun ist es aber so, dass diese Freiheiten mitnichten auf Modulpools beschränkt sind. Sie können auch in ausführbaren Programmen jederzeit mit der Anweisung CALL SCREEN Dynpros aufrufen und damit die Programmsteuerung dem Dynprozessor übergeben. Wenn Sie auf keines der Reporting-Ereignisse außer START-OF-SELECTION reagieren und dort ein Einstiegsdynpro aufrufen, verhält sich ein ausführbares Programm letztlich genauso wie ein Modulpool, mit der Ausnahme, dass es über eine Namens eingabe gestartet werden kann. Wir haben diese Technik bereits in unserem ersten Programm S\_FIRST\_PROGRAM in Abschnitt 2.3 verwendet. Wenn Sie unbedingt wollen, können Sie die Dynpros von ausführbaren Programmen sogar mit Transaktionscodes verbinden. Dann werden ausführbare Programme unabhängig von ihrem Typ exakt wie Modulpools ausgeführt, und es treten keine Reporting-Ereignisse auf.

Wie ausführbare Programme Dynpros und Dialogmodule enthalten können, können Modulpools auch Selektionsbilder und Listen sowie die zugehörigen Ereignisblöcke enthalten. Deren Prozesse müssen dann aber mit den Anweisungen CALL SELECTION-SCREEN und LEAVE TO LIST-PROCESSING aufgerufen werden, da keine automatischen Aufrufe erfolgen. Die einzige Einschränkung von Modulpools gegenüber Reports sind also die fehlenden Reporting-Ereignisse und die Möglichkeit, sie über SUBMIT zu starten.

Warum gibt es dann überhaupt diese Trennung in ausführbare Programme und Modulpools? Die Antwort liegt im klassischen ABAP-Programmiermodell, welches fein säuberlich zwischen Reportprogrammierung und Transaktions- bzw. Dialogprogrammierung unterscheidet. Falls Sie bereits Erfahrungen in der ABAP-Programmierung früherer Basis-Releases gesamt

**Report/  
Transaktion**

melt haben und mit entsprechender Literatur vertraut sind, so wird Ihnen diese Trennung insbesondere auch in Schulung und Dokumentation in guter Erinnerung sein.

Wir sprachen damals von Reports und von Dialogprogrammen. Reports dienten dem lesenden Zugriff auf Datenbanken, Dialogprogramme bzw. Transaktionen dem ändernden Zugriff. Für jede dieser beiden Aufgaben war ein spezialisierter Programmtyp vorgesehen. Der Reportprogrammierer hat ausschließlich ausführbare Programme angelegt, sich weitgehend an deren Ablauf gehalten und sich auf deren Bildschirmbilder beschränkt. Der Dialogprogrammierer hat ausschließlich Modulpools mit Dynpros und zugehörigen Transaktionscodes angelegt. Die schon immer vorhandenen gemeinsamen technischen Grundlagen aller ABAP-Programmtypen wurden in diesen Programmiermodellen einfach nicht benötigt. Verbindungen zwischen den beiden Programmier Techniken, wie z. B. der Aufruf eines Dynpros in einem ausführbaren Programm, wurden eher wie Ausnahmen behandelt, und der Aufruf eines Selektionsbilds außerhalb ausführbarer Programme war vor Release 4.0 schlichtweg nicht möglich.

Im Zeitalter von ABAP Objects haben wir diese getrennte Sicht durch die in diesem Kapitel vorgestellte Gesamtsicht ersetzt. Es ist Ihre Sache als Anwendungsentwickler, gemäß den Anforderungen an Ihr Programm zu entscheiden, welche Art der Programmausführung Sie wählen. Es gibt keine Vorschrift, die lesenden Datenbankzugriffe auf ausführbare Programme beschränkt oder ändernde Datenbankzugriffe auf Modulpools. Genauso wenig hat es noch Sinn, den Einsatz von Dynpros auf Modulpools zu beschränken. Wenn Sie die neuen Programmier Techniken mit ABAP Objects und GUI-Controls in ausführbaren Programmen einsetzen wollen, müssen Sie Dynpros verwenden.

Der Einsatz von Modulpools ist immer dann sinnvoll, wenn Sie keine der Automatismen eines ausführbaren Programms benötigen und Ihr Programm nur über einen Transaktionscode starten wollen. Ausführbare Programme haben den Vorteil, direkt startbar zu sein, auch wenn man keines der übrigen Reporting-Ereignisse benötigt. Beachten Sie aber, dass die Ausführung einer Transaktion im SAP-System einfacher mit einer Berechtigungsprüfung verknüpft werden kann, als die Ausführung eines ausführbaren Programms.

**Funktionsgruppe** Zum Abschluss dieser Diskussion sei noch darauf hingewiesen, dass sämtliche Bildschirmarten genauso wie in den beiden hier diskutierten Programmen auch in Funktionsgruppen verwendet werden können. Sie können in einem Funktionsbaustein beispielsweise mit CALL SCREEN oder CALL

SELECTION-SCREEN Dynpros oder Selektionsbilder aufrufen und deren Ereignisse in der Funktionsgruppe behandeln. Die Programmierung von Bildschirmbildern ist also nicht nur auf ausführbare Programme oder Modulpools beschränkt. Sie können vollständige Dialoge in Funktionsgruppen kapseln und aus beliebigen ABAP-Programmen aufrufen. Classpools unterstützen aber keines der klassischen Bildschirmbilder.

### **3.3.6 Programme intern aufrufen**

Aus Benutzersicht gibt es zwei Möglichkeiten, Programme zu starten, nämlich die direkte Ausführung über den Programmnamen bei ausführbaren Programmen und die Auswahl eines Transaktionscodes bei Modulpools. Diese beiden Arten des Programmaufrufs sind auch in bereits laufenden ABAP-Programmen durchführbar. Wir haben die betreffende Anweisung SUBMIT für ausführbare Programme schon mehrfach erwähnt. Die entsprechenden Anweisungen für Transaktionen sind LEAVE TO TRANSACTION und CALL TRANSACTION.

Aufrufe von Programmen aus anderen ABAP-Programmen heraus können wir danach unterscheiden, ob das aufrufende Programm vollständig verlassen wird oder ob das aufgerufene Programm in das aufrufende Programm eingebettet wird.

#### **Vollständiges Verlassen des aufrufenden Programms**

Sie verlassen das aufrufende Programm vollständig, wenn Sie ein anderes Programm mit

**SUBMIT** prog.

oder

**LEAVE TO TRANSACTION** ta.

aufrufen. Bei der Anweisung SUBMIT wird das aufrufende Programm mit allen seinen Daten aus dem Speicher gelöscht. Nach Beendigung des aufgerufenen Programms kehrt die Programmausführung an die Stelle zurück, von der das aufrufende Programm gestartet wurde. Diese Stelle kann ein vorhergehendes Programm einer Aufrufkette sein.

Bei der Anweisung LEAVE TO TRANSACTION werden außer dem aufrufenden Programm alle vorhergehenden Programme einer Aufrufkette mit ihren Daten aus dem Speicher gelöscht. Nach Beendigung des aufgerufenen Programms kehrt die Programmausführung an die Stelle zurück, von der das erste Programm der Aufrufkette gestartet wurde.

**LEAVE** Die Anweisung LEAVE wird uns auch noch in anderen Zusammenhängen begegnen. Mit einer LEAVE-Anweisung wird ein Programmkontext vollständig verlassen, und die Ausführung kehrt nicht hinter diese Stelle zurück. LEAVE kann zwar ohne Zusätze verwendet werden, wir empfehlen aber, LEAVE nur mit einem Zusatz wie hier TO TRANSACTION zu verwenden. Ansonsten kann oft nicht eindeutig vorhergesagt werden, wohin die Programmausführung zur Laufzeit tatsächlich verzweigt, da das Verhalten vom Aufrufmodus des aktuellen Programms abhängt.

### **Einbettung des aufgerufenen Programms in das aufrufende Programm**

Sie verlassen das aufrufende Programm temporär, wenn Sie ein anderes Programm mit

**SUBMIT prog AND RETURN.**

oder

**CALL TRANSACTION ta.**

aufrufen. Bei beiden Anweisungen bleibt das aufrufende Programm mit seinen Daten erhalten. Nach Beendigung des aufgerufenen Programms kehrt die Programmausführung direkt hinter die Aufrufstelle zurück.

Auch die Anweisung CALL werden wir noch oft wieder sehen. Die Anweisung CALL ist die wichtigste Anweisung, um in ABAP einen Programmkontext temporär zu verlassen, um eine andere Programmeinheit auszuführen. Nach ordnungsgemäßer Beendigung der aufgerufenen Einheit wird hinter die aufrufende Stelle zurückgekehrt.

### **3.3.7 ABAP-Programme beenden**

Die Ausführung eines ABAP-Programms ist immer dann beendet, wenn der zugehörige Prozess in der Laufzeitumgebung beendet wird. Aus Sicht des ABAP-Programmierers ist dies nach der Beendigung des letzten von der Laufzeitumgebung getriggerten Verarbeitungsblocks.

Sie können ABAP-Programme aber jederzeit auch programmgesteuert mit der Anweisung

**LEAVE PROGRAM.**

beenden. Wie im vorhergehenden Abschnitt gezeigt, beenden auch die Anweisung LEAVE TO TRANSACTION und die Anweisung SUBMIT ohne den Zusatz AND RETURN ein ABAP-Programm.

### 3.4 Speicherorganisation von ABAP-Programmen

Zum Abschluss unserer Einführung in die Grundlagen der ABAP-Programmierung wollen wir uns noch kurz mit der Speicherorganisation von ABAP-Programmen beschäftigen. In diesem Abschnitt lernen Sie noch einige Begriffe kennen, die Ihr Verständnis der Ausführung von ABAP-Programmen vertiefen sollen.

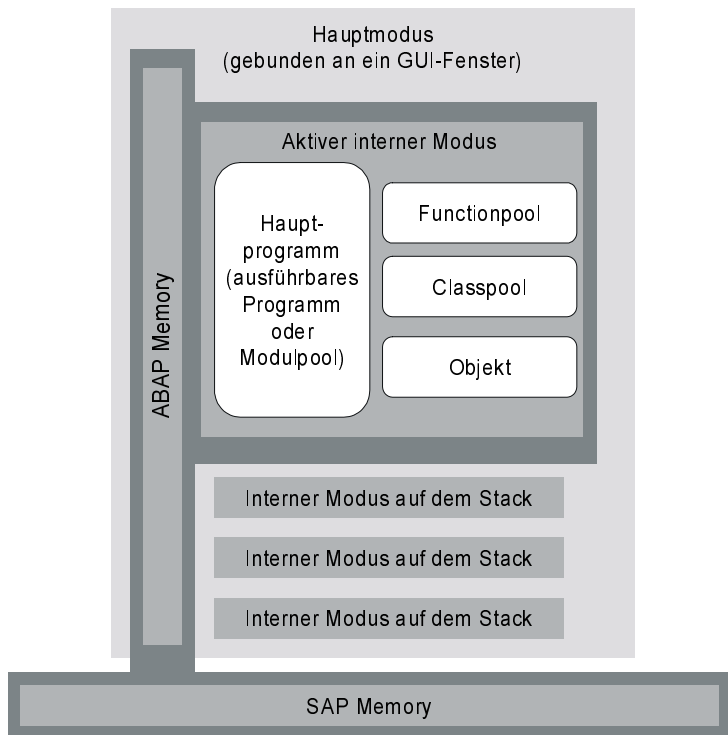


Abbildung 3.12 Speicherorganisation von ABAP-Programmen

Abbildung 3.12 zeigt eine Übersicht über die Speicherorganisation, deren Aufbau wir im Folgenden erläutern.

#### Hauptmodus

Wenn Sie sich an einem SAP-System anmelden, öffnen Sie auf dem Applikationsserver einen Hauptmodus, auch externer Modus genannt. Sie können über **System · Erzeugen Modus** innerhalb einer Anmeldung bis zu fünf weitere Modi öffnen. Jeder dieser sechs Modi ist mit einem eigenen

Bildschirmfenster im SAP GUI verknüpft und belegt einen eigenen Speicherbereich auf dem Applikationsserver. Untereinander verhalten sich diese Modi fast wie unabhängige SAP-Anmeldungen.

### **Interner Modus**

Die Ausführung eines ausführbaren ABAP-Programms oder eines Modul-pools öffnet einen internen Modus im Hauptmodus. Der Speicherbereich des ABAP-Programms ist Teil des Speicherbereichs des Hauptmodus. Die Daten und Objekte eines Programms leben in seinem internen Modus. In einem Hauptmodus ist aber immer nur der interne Modus des gerade ablaufenden Programms aktiv vorhanden. Alle Speicherinhalte von vorhergehenden Programmen einer Aufrufkette sind auf einem Stack gespeichert.

### **Aufrufketten**

Wie wir oben gesehen haben, können ABAP-Programme andere ABAP-Programme aufrufen, wobei der Aufrufer entweder vollständig verlassen oder nach Beendigung des aufgerufenen Programms zu ihm zurückgekehrt wird. Wenn nach der Ausführung des aufgerufenen Programms nicht in das vorherige Programm zurückgekehrt wird, ersetzt der interne Modus des aufgerufenen Programms den internen Modus des aufrufenden Programms, und dessen Speicherinhalt wird vollständig gelöscht. Wird aber in das aufrufende Programm zurückgekehrt, wird der Zustand seines Modus während der Ausführung des aufgerufenen Programms auf einem Stack erhalten. Ein Programm bildet mit seinen Vorgängern auf dem Stack eine Aufrufkette. Zwischen den Programmen einer Aufrufkette können Daten über das ABAP Memory übergeben werden.

### **ABAP Memory**

Im Speicherbereich jedes Hauptmodus gibt es einen Bereich namens ABAP Memory, auf den die Programme im internen Modus mit EXPORT TO MEMORY und IMPORT FROM MEMORY zugreifen können. Daten innerhalb des ABAP Memory bleiben über eine Abfolge von Programmaufrufen hinweg erhalten. Um Daten an ein aufgerufenes Programm zu übergeben, können diese vor dem Programmaufruf mit der Anweisung EXPORT in das ABAP Memory gestellt werden. Dann ersetzt der interne Modus des aufgerufenen Programms den internen Modus des aufrufenden Programms, und die Daten können mit der Anweisung IMPORT aus dem ABAP Memory gelesen werden. Genauso können bei der Rückkehr ins aufrufende Programm Daten übergeben werden (siehe auch Abschnitt 8.3.1).

## **SAP Memory**

Das SAP Memory ist ein Speicherbereich, auf den sämtliche Modi einer SAP-Anmeldung gemeinsamen Zugriff haben. In ABAP-Programmen kann mit den Befehlen SET PARAMETER und GET PARAMETER auf die so genannten SPA/GPA-Parameter des SAP Memory zugegriffen werden. Eingabefelder auf Dynpros können mit solchen Parametern verknüpft und dadurch vorgelegt werden.

## **Programme laden**

Die Ausführung oder der Aufruf eines ABAP-Programms lädt eine Instanz des Programms in den internen Modus. Das erste Programm eines internen Modus ist dessen Hauptprogramm. Wenn im Hauptprogramm eines internen Modus eine Prozedur eines anderen ABAP-Programms aufgerufen oder eine öffentliche Komponente einer globalen Klasse eines Classpools angesprochen wird, wird eine Instanz des zugehörigen ABAP-Programms in den internen Modus geladen und bleibt mit ihren Daten und Objekten dort solange vorhanden, bis das Hauptprogramm beendet wird. Auch die hinzugeladenen Programme können weitere Programme laden. Bei jedem Laden einer Instanz wird bei allen Programmen außer bei Classpools das Ereignis LOAD-OF-PROGRAM ausgelöst.

## **Objekte erzeugen**

Beim Erzeugen von Objekten über CREATE OBJECT in ABAP Objects werden Instanzen von Klassen in den internen Modus geladen. Alle Programme und Objekte eines internen Modus können auf die Instanzen des gleichen internen Modus zugreifen. Es können also Referenzen auf Objekte innerhalb des internen Modus an externe Prozeduren (Unterprogramme, Funktionsbausteine und Methoden) übergeben werden.

Es gibt aber keine Speicherbereiche für transaktionsübergreifende Objekte auf dem Applikationsserver. Referenzen können nicht im ABAP Memory oder SAP Memory gespeichert werden. Persistente Objekte in der Datenbank müssen durch Persistenzdienste behandelt werden, die nicht zum Sprachumfang von ABAP gehören.

**Persistenz**