Universität Hamburg
Informatik / Datenbanken und Informationssysteme

DIPLOMARBEIT

9. Juni 1997

Eine polymorph typisierte Schnittstelle der Sprache Tycoon zum System SAP R/3

Betreuer: Prof.Dr.J.W.Schmidt
Prof.Dr.rer.pol.D.Preßmar

Vorgelegt von Sebastian Lutz
Heussweg 71
20255 Hamburg

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Ziel der Arbeit . . . 6
- 1.2 Gliederung der Arbeit . . . 6
- 1.3 Spezielle Notationen und graphische Modelle . . . 8

2 Tl – eine Programmiersprache zur Systemintegration **11**
- 2.1 Eigenschaften der Sprache Tl und ihre semantischen Objekte . . . 12
  - 2.1.1 Semantische Objekte . . . 15
  - 2.1.2 Persistenz . . . 15
  - 2.1.3 Mobilität . . . 16
- 2.2 Integration externer Systeme . . . 17
  - 2.2.1 Client/Server-Kommunikation . . . 18
  - 2.2.2 Externe Bibliotheken . . . 18
  - 2.2.3 Integrative Schranken . . . 20
- 2.3 Ausgewählte Systemintegrationen . . . 20
  - 2.3.1 SQL . . . 21
  - 2.3.2 Stp/OMT . . . 22
  - 2.3.3 NFS . . . 23
  - 2.3.4 INQUERY . . . 24
  - 2.3.5 C++-Gateway . . . 24
  - 2.3.6 Zusammenfassung . . . 25

3 Integrationsmöglichkeiten des SAP R/3 Systems **27**
- 3.1 Modellarchitektur des Systems . . . 28
  - 3.1.1 Dienste und Prozeßarchitektur der R/3-Middleware . . . 28
  - 3.1.2 Semantische Objekte . . . 30
  - 3.1.3 Offene Schnittstellen . . . 33
- 3.2 Grundlagen der Programmierung für Funktionsbausteine . . . 38
  - 3.2.1 Die Workbench . . . 38
  - 3.2.2 Das Datenmodell . . . 39
  - 3.2.3 Die Programmiersprache ABAP/4 . . . 45
  - 3.2.4 Programmierung mit semantischen Objekten des SAP R/3 . . . 49
  - 3.2.5 Transaktionen in SAP R/3 . . . 53
- 3.3 Zusammenfassung . . . 55

4 Die SAP R/3-Funktionsbibliothek **57**
- 4.1 Organisation und Verwaltung der Funktionsbausteine . . . 58
  - 4.1.1 Schnittstellendefinition . . . 60
  - 4.1.2 Aufruf- und Parameterübergabekonventionen . . . 63
- 4.2 Die RFC-Bibliothek des SAP R/3 für externe Systeme . . . 69

4.2.1 Ein Modell für externe RFC-Funktionen . . . 69
4.2.2 Entwicklungsunterstützung durch die Workbench . . . 71
4.2.3 Der aktive Funktionsaufruf . . . 71
4.2.4 Der passive Funktionsaufruf . . . 74
4.2.5 Rückrufe in Funktionsaufrufen . . . 75
4.2.6 Besonderheiten beim transaktionalen RFC . . . 76
4.3 Zusammenfassung . . . 78

5 Konzepte zur Integration des SAP R/3 in Tycoon 79
5.1 Vergleich der Systeme und Anforderungen an eine Tycoon-SAP R/3-Schnittstelle . . . 79
5.2 Softwarearchitektur des Integrations-Gateways . . . 82
5.2.1 Parameter und ihre Bindungen . . . 84
5.2.2 Unterstützung von Persistenz und Mobilität . . . 89
5.2.3 Applikationschnittstellen . . . 91
5.2.4 Behandlung von Fehlersituationen durch Ausnahmen . . . 95
5.3 Erweiterung um statische Typprüfung . . . 96
5.3.1 Typ- und Signaturkonstruktoren . . . 97
5.3.2 Strukturtypen für Tabellen-Parameter . . . 98
5.3.3 Abbildung von Funktionen variabler Parameterzahl in Tl . . . 99

6 Zusammenfassung der Ergebnisse und Ausblick 101
6.1 Strukturen und Schnittstellen des SAP R/3 . . . 102
6.2 Integration über ein vierschichtiges Gateway . . . 102
6.3 Ausblick und offene Probleme . . . 103

A Glossar 105

Abbildungsverzeichnis

1.1 Architektur eines Informationssystems . . . 3
1.2 Modellierung ist eine Frage der Skalierung . . . 4
1.3 Orthogonale Tycoon-Systemeigenschaften und -Freiheitsgrade . . . 6

2.1 Tycoon erlaubt eine einheitliche Behandlung aller Objekttypen . . . 17
2.2 Modulhierarchie der Tycoon-SQL-Schnittstelle . . . 21
2.3 Funktionsverschachtelung beim Öffnen einer Tabelle . . . 22
2.4 NFS-Anbindung aus [Gor95] . . . 23

3.1 Grobarchitektur des SAP R/3 Systems aus [BEG96] . . . 28
3.2 Dienstehierarchie des SAP R/3-Systems . . . 29
3.3 SAP R/3 Repository (Darstellung aus [WHSH95]) . . . 30
3.4 Semantische Objekte des SAP R/3 . . . 31
3.5 Ereignispfade der SAP-Transaktionen-Verarbeitung . . . 32
3.6 Offentheit von SAP R/3 (Abbildung aus [SAP96c]) . . . 33
3.7 Die sieben Schichten des SNA-Modells und deren Zuordnung zum OSI-Modell (→[Ste92]) 35
3.8 Unidirektionale CPI-C-Komunikation . . . 35
3.9 Die Workbench (→[SAP94a]): Werkzeuge, Verwaltung und Objekte . . . 39
3.10 Das SAP Data Dictionary . . . 40
3.11 Die SAP R/3-Typhierarchie . . . 42
3.12 Menge der externen Typen . . . 43
3.13 Menge der ABAP/4 Typen . . . 44
3.14 Zusammenhang der verschiedenen Datentypebenen des SAP R/3 . . . 45
3.15 Eingabedaten der Batch Input-Steuerung für SAP-Transaktionen . . . 52

4.1 Struktur der Funktionsbiblithek . . . 58
4.2 Verwaltung für Funktionsbausteine . . . 59
4.3 Import-, Export- und Changing-Parameter von Funktionsbausteinen . . . 61
4.4 Tabellen-Parameter und Ausnahmen von Funktionsbausteinen . . . 62
4.5 Vergleich des synchronen mit dem asynchronen RFC . . . 65
4.6 Objektmodell für externe RFC-Funktionen . . . 70
4.7 Programmgenerierung mittels des RFC-Generators . . . 71
4.8 Zustandsmodell des RFC für einen aktiven Funktionsaufruf . . . 72
4.9 Logon-Information eines RFC-Aufrufs . . . 73
4.10 Zustandsmodell des RFC für einen passiven Funktionsaufruf . . . 74
4.11 Zustandsübergänge bei einem RFC-Rückruf . . . 76
4.12 Transaktionaler Aufruf einer Funktion . . . 77
4.13 Installation der Hilfsfunktionen für die Transaktionsabwicklung . . . 78

5.1 Architektur der Tycoon-SAP R/3-Integrationsschnittstelle . . . 80
5.2 Vier-Schichtenmodell des Gateways . . . 82

5.3 Strukturen der nichttrivalen ABAP/4-Basistypeinbettungen 85
5.4 Erweitertes Objektmodell für externe RFC-Funktionen 86
5.5 Parameter- und Tabellenstruktur für RFC-Aufrufe 87
5.6 Polymorphes Marshalling der Aktualparameter 88
5.7 Unterstützung der Pseudo-Persistenz bzw. -Mobilität 89
5.8 Dynamische Abläufe der Funktionen `stress()` und `relax()` 90
5.9 Hierarchie möglicher Ausnahmen 96
5.10 Ausnahmefilter der Schicht zwei 97
5.11 Abstraktes Objektmodell der Gateway-Schicht vier 100

Tabellenverzeichnis

2.1 Eigenschaften verschiedener Integrationsschnittstellen des Tycoon-Systems 25

3.1 Zustände eines ABAP/4-Programms im Vergleich 46
3.2 Vergleich der Methoden des Batch Input 53

4.1 Eigenschaften verschiedener RFC-Aufrufmodi 68
4.2 Zusammenhang der ABAP/4-Programmierung mit dem Transaktionensystem 68

5.1 Abbildung der ABAP/4-Basistypen nach Tl 85

Kapitel 1

Einleitung

Mit zunehmder Globalisierung der Märkte und sich verkürzenden Innovationszyklen unterliegen Unternehmen in der heutigen Zeit einem stark anwachsenden Wettbewerbsdruck. Zentrale Erfolgsfaktoren, die ein langfristiges Überleben garantieren, sind eine Reduzierung der betrieblichen Kosten und die Flexibilität Markttendenzen zu folgen oder neue zu setzen. Denn weniger sind absolute Maßstäbe die Triebfeder, sondern der Kunde mit seinen Anforderungen an Funktionalität, Qualität, Lieferzeit und Preis bestimmt über Erfolg oder Mißerfolg [BEG96, DM92].

Ein Werkzeug den angegebenen inner- und außerbetrieblichen Anforderungen gerecht zu werden ist die elektronische Datenverarbeitung. Im Vergleich mit frühen Anwendungen im betrieblichen Bereich, die sie für spezialisierte Aufgaben, wie Textverarbeitung oder Fakturierung, zum Außenseiter machten, verkürzt ihre Effizienz Verzögerungen in den Durchlaufzeiten entlang von Wertschöpfungsketten und Reaktionszeiten bei deren Rekonfiguration. Andererseits ist es auch die EDV, die die Einplanung individueller Kundenwünsche in die Produktion beherrschbar macht.

Aus Kosten- und Organisationsgründen entscheiden sich heute viele Unternehmen für Standardsoftware-Produkte. Während bei Firmenprojekten zur Entwicklung betriebswirtschaftlicher Programme hohe Entwicklungskosten für Analyse, Design, Implementierung und Einführung veranschlagt werden müssen, wobei das Risiko für fehlerhafte Kostenvoranschläge und ungenaue Zeitabschätzungen dieser vier zentralen Projektphasen immens ist, werben die Produzenten von Standardsoftware mit Unternehmensreferenzmodellen und einfachen Vorgehensstrategieen (→[BEG96] Seite 235ff). Diese ersetzen die Analyse- und Designphasen durch eine gut überschaubare Abbildung vorhandener, neugebildeter oder radikal erneuerter[1] Geschäftsprozesse auf das Referenzmodell [Sch95]. Die Implementation wird durch ein Einstellen der System- und Anwendungsparameter der vordefinierten Funktionen der Standardsoftware abgelöst, so daß nur die Einführungsphase bleibt.

Konzepte der SAP R/3 Standardsoftware

SAP war der erste Anbieter für betriebliche Standardsoftware, so daß sie heute weltweiter Marktführer[2] ist. Die Strategie der SAP ist es ein vollintegriertes Standardpaket anzubieten, das die Abläufe, Funktionen und Daten der meisten Unternehmen abdeckt. Dazu wurde mit R/3 eine Architektur entwickelt, die eine „Middleware“(vgl. Abschnitt3.1) als Basis hat und darauf aufsetzende, zum Teil ineinander verzahnte Standardmodule für betriebliche Anwendungen bestimmt.

Die wichtigsten informationstechnischen Konzepte des Systems R/3 sind:

Relationenmodell Alle Daten des SAP R/3-Systems sind in Tabellen abgelegt. Dies betrifft nicht nur Anwendungsdaten, sondern auch das Data Dictionary, Programme, SAP-Transaktionen und sonstige semantischen Konstrukte. Als Vorteil ergibt sich damit, daß für SAP-Anwendungen

[1] Häufig als „Business Process Reengineering“ bezeichnet

[2] Konkurrenzprodukte der Firmen Baan, Oracle und Peoplesoft sind noch neu und decken mit ihrer Funktionalität teilweise nur Teilbereiche des Umfangs von SAP R/3 ab [BBC+96].

die gebräuchlichen semantischen Erweiterungen des Relationenmodells (→[Dat95]), wie z.B. das Entity Relationship-Modell, zur Modellierung genutzt werden können. Außerdem können Integritätsbedingungen der Anwendungen über Fremdschlüssel ausgedrückt werden.

4GL-Sprache Die Programmiersprache des SAP R/3-Systems ist ABAP/4. Sie zeichnet sich durch hochgradig spezialisierte Befehle, die fest in die Systemumgebung eingebunden sind, aus. Durch die Portabilität der R/3-Middleware sind alle ABAP/4-Programme auch portabel.

Client/Server-Modell Verteilung wird in R/3 durch Client/Server-Beziehungen von ABAP/4-Programmen erreicht. Dabei können die als Funktionsbausteine (vgl. Abschnitt 4) ausgezeichneten ABAP/4-Programme ortstransparent aufgerufen werden, um Datenbank-, Berichts- oder sonstige Dienste durchzuführen.

Integration Alle Funktionen des SAP R/3 sind, abgesehen von einem Kern der Middleware, der in C geschrieben ist und an die jeweilige Hardware angepaßt wird, selbst mit den Mitteln des R/3-Systems entwickelt und realisiert. Das betrifft nicht nur die betriebswirtschaftlichen Funktionen, sondern auch große Teile der Entwicklungsumgebung und anderer Komponenten.

Customizing [3] Die betriebswirtschaftlichen Anwendungen sind in einer solchen Allgemeinheit verfaßt, daß sie „fast alle“ Bedürfnisse von „fast allen“ Unternehmen abdecken. Um jedoch ein spezielles Unternehmen in R/3 abzubilden, müssen die konkreten Daten des Unternehmens im R/3-System voreingestellt, eben gecustomized, werden. Dabei werden z.B. die Darstellungen des Datums oder der Währung als länderspezifische Einstellungen oder die benötigten Daten, Funktionen und Prozesse des SAP-Referenzmodells für die Darstellung der unternehmensspezifischen Daten, Funktionen und Prozesse eingestellt.

Alle übrigen Dienste, Leistungen und Werkzeuge des SAP R/3 sind auf der Basis dieser fünf Konzepte entwickelt.

SAP verfolgt damit daten- und softwaretechnisch einen modulorientierten Ansatz. Insbesondere sind Daten in Tabellen abgespeichert, die ihre Beziehungen, wie im Relationenmodell, über Assoziationen herstellen, während Programme in Module strukturiert sind, die im System R/3 *Funktionsbausteine* heißen, und sich gegenseitig über klar definierte Schnittstellen, ähnlich dem Modell des abstrakten Datentyps [BG84], aufrufen können. Die Merkmale dieser assoziationsbasierten Modellierung von Daten und Funktionen sind klar abgegrenzte Sichtbarkeitsbereiche einerseits, aber auch transitive Abhängigkeiten, die bei Änderungen in Tabellen oder Funktionsbausteinen zu einem nicht unerheblichen „Overhead“ bei dem Nachvollziehen der Tabellenassoziationen oder Funktionsbausteinschnittstellen führen können.

Systemtechnisch ist die SAP-Software ein verteiltes Client/Server-System, das allen aufsetzenden Komponenten Portabilität, offene Schnittstellen zu anderen Systemen und Integration innerhalb der Grenzen der Middleware gewährleistet. SAP versucht durch das Hinzufügen neuer Techniken und die Übernahme verbreiteter Standards neuen Anforderungen gerecht zu werden.

Es ergibt sich die Frage, durch welche Charakteristika sich betriebliche Informationssysteme im allgemeinen auszeichnen und ob eventuell andere Ansätze oder Erweiterungen des SAP-Systems Vorteile bringen oder langfristig zu einer Konsolidierung der inzwischen sehr umfangreichen System- und Anwendungsarchitektur führen.

Eigenschaften moderner Informationssysteme

Moderne Informationssysteme profitieren unter anderem aus den drei Bereichen der offenen verteilten Systeme [Nut92, CDK94], der Datenbanken und der Programmiersprachen [MS97], wobei die letzteren beiden im Rahmen der Datenbankprogrammiersprachen [PCWT96] bereits gegenseitigen Einfluß nehmen. In der Theorie der offenen verteilten Systeme werden die Grundlagen der Protokolle, Schnittstellen und Verfahren von Computer-Anwendungen aus einer technisch dominierten Sicht untersucht.

[3] Aussprache, wie der englische gleichgeschriebene Begriff.

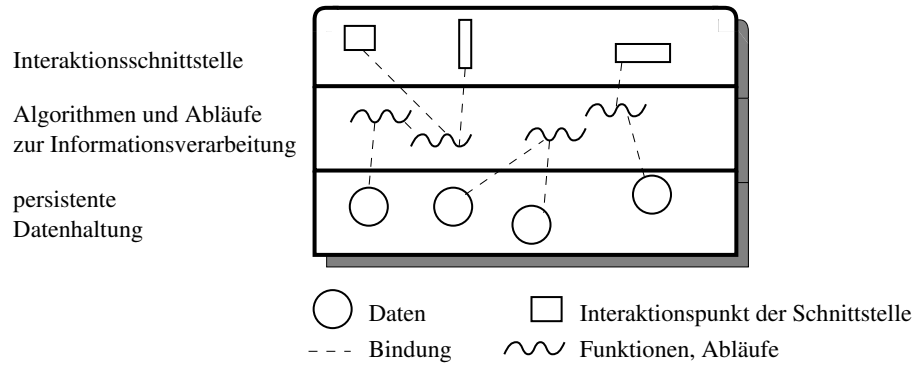


Abbildung 1.1: Architektur eines Informationssystems

Entsprechend [Nut92] können offene Systeme in den drei Dimensionen Integration, Interoperabilität und Portabilität beschrieben werden:

Interoperabilität ist die Fähigkeit zweier Systeme, zwischen mehreren ihrer Komponenten gleichen Levels Informationen auszutauschen.

Integration ist die Eigenschaft einer Menge von Anwendungen, Komponenten oder Systemen eine gemeinsame konsistente Schnittstelle zu besitzen.

Portabilität bedeutet für ein System, in mehreren Umgebungen lauffähig zu sein.

In [MS97] wird das in Abbildung 1.1 dargestellte allgemeine Modell für Informationssysteme vorgeschlagen. Demnach besteht ein Informationssystem aus den drei Komponenten:

Interaktions(Zugriffs-)schnittstelle Sie ist die Schnittstelle für Aktivitäten der Umgebung. Das kann, wie im Fall von SAP R/3, eine interaktive Benutzerschnittstelle sein. Möglich ist aber auch eine Programmierschnittstelle, die den Zugriff kapselnder Systeme, Workflows oder allgemeiner Aktivitäten auf das Informationssystem ermöglicht.

Algorithmen zur Informationsverarbeitung und Integritätssicherung Hier werden die Daten der Objektspeicherebene algorithmisch verdichtet oder neu verknüpft. Dadurch findet eine semantische Bereicherung statt, so daß im Sinn des IS Informationen[4] vorliegen. Desweiteren werden Integritätsregeln durch die Algorithmen zugesichert, die auf der Ebene der Datenbank nicht geprüft werden können, weil sie von der semantischen Bereicherung der Algorithmen dieser Ebene profitieren.

Persistenter Objektspeicher Er dient als Repository für Daten. Auf seiner Ebene werden Daten abgelegt und gelesen. Die üblichen Datenbankdienste sichern die Integrität indem sie konkurrierenden Zugriff nebenläufiger Prozesse der übergeordneten Verarbeitungsebene serialisieren.

Als grundlegende Eigenschaften von Informationssystemen werden Quantität, Persistenz, Reaktivität und Integrität, kurz *QPRI*, angegeben:

Quantität/Regularität bedeutet das Vorhandensein eines regulär strukturierten umfangreichen Datenbestandes. Viele Daten sind durch dasselbe Schema beschrieben.

Persistenz heißt, daß die Daten langlebig im Sinn des Systems sind. Insbesondere überleben sie ihren Erzeugerprozess.

[4] In [BG82] Seite 3ff+44ff. wird zwischen Information und Nachricht unterschieden. Die Information wird durch eine Interpretationsvorschrift aus der Nachricht gewonnen. Im Kontext der Informationssysteme entsprechen Nachrichten den Daten und die Interpretationsvorschrift den Algorithmen

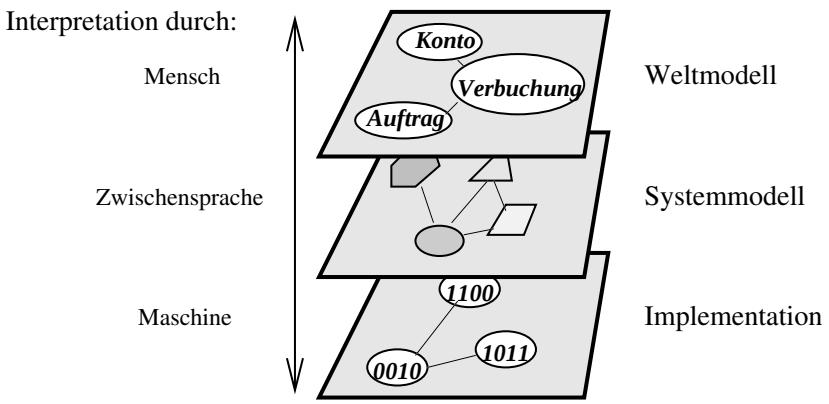


Abbildung 1.2: Modellierung ist eine Frage der Skalierung

Reaktivität/Interaktivität gibt den Umfang der Interaktionsmöglichkeiten mit dem System an. Ein Informationssystem muß möglichst viele Interaktionsmöglichkeiten für möglichst viele Systeme haben.

Integrität Ein Informationssystem birgt eigene Integritätsregeln. Dazu gehören konsistenzerhaltene Regeln der Datenbeziehungen, Regeln für zeitliche Präferenzen und semantische Integrität der Aktivitäten eines Informationssystems und das korrekte Protokoll für Interaktion mit der Informationssystemschnittstelle.

Die QPRI-Eigenschaften sind eine gute extensionale Charakterisierung, enthalten aber noch keine Kriterien, die das dynamische Verhalten über die Lebensdauer, die Flexibilität bezüglich der Anwendungsdomäne und Modellierungsfähigkeiten oder den konkreten Bezug zu anderen Systemen herstellen. Deshalb sollen die erweiterten Eigenschaften Evolution, Skalierung, Verteilung und Offenheit im folgenden beschrieben werden:

Evolution Informationssysteme speichern Daten und leisten gewisse Dienste. Dabei wächst mit der Zeit die Menge der Daten. Zusätzlich werden aber auch neue Anforderungen an ein Informationssystem gestellt, die dann in Dienste umgesetzt werden. Das heißt Informationssysteme entwickeln sich. Dieser Prozeß kann so weit gehen, daß ein Wechsel zu einem größeren System vollzogen werden muß, weil Systembeschränkungen kein weiteres Wachstum mehr zulassen. Ein Fall der häufig mit dem Aufwand der Übernahme der Daten in das neue System verbunden ist, wie es speziell im Zusammenhang mit der Neueinführung eines SAP-Systems in [Men96] beschrieben wird. Allgemein muß aber jedes Informationssystem befähigt sein, über eine rein quantitative Änderung seiner Datenbestände hinaus neue Dienste oder gewisse Änderungen der Verarbeitung in der Informationsverarbeitungsebene zu erlauben.

Skalierung Die Anforderungen an Informationssysteme sind nicht nur anwendungsbezogen unterschiedlich, sondern auch hinsichtlich der Problemgröße gibt es starke Schwankungen in Komplexität und Umfang der Daten, Funktionen und Abläufe. Hinzu kommt, daß sich durch die zuvor beschriebene Evolution eines Informationssystems diese Größen mit der Zeit noch verändern. Daher sollten die Modellierungsmöglichkeiten, gegeben durch das Typsystem oder die Sprache eines Informationssystems, möglichst vielfältig und polymorph sein. Die Beschränkung auf nur ein Datenmodell, wie es das Relationenmodell ist, führt zu Einschränkungen des Anwendungsfeldes oder kann, was der schlimmere Fall ist, nachträglich zu übermäßigem Aufwand führen. Dieses rührt daher, daß Beziehungen der Relationen bei Änderungen im Modell transitiv über alle beteiligten Fremdschlüsselverbindungen oder Assoziationen hinweg nachvollzogen werden müssen. Wie in Abbildung 1.2 dargestellt, ist deshalb eine weltmodellnahe Zwischensprache zur Modellierung ein wichtiges Kriterium für Informationsysteme. Moderne 4GL-Datenbankprogrammiersprachen sind in der Regel mit leistungsfähigen, an der

Anwendungsdomäne orientierten und in ein Informations- oder Datenbanksystem vollständig integrierten Befehlen ausgestattet. Jedoch bietet dieser Ansatz nur extrem schlechte Skalierungseigenschaften dadurch, daß weder die Möglichkeit besteht den durch die Integration in das System bedingten „Overhead“ zu umgehen [MS93], noch eine dem Problem angepaßte Modellierung zu wählen. Es können damit nur solche Probleme modelliert werden, die durch die Art und Extension der komplexen Befehle vorgedacht sind.

Verteilung Große Anwendungen laufen heutzutage nicht mehr lokal ab. Firmen tendieren dazu Intranetze zu installieren und nahezu alle Universitäten und Forschungseinrichtungen der Welt sind im Internet miteinander verbunden. Dabei ist das zentralistische Modell des Großrechners mit angeschlossenen Terminals weitgehend überholt. Es zeichnet sich durch Standards, wie CORBA, OLE/COM und DSOM (→[OHE96b]), ab, daß zukünftige Anwendungen dezentral aber in Kooperation ablaufen. Dabei wird nicht die herkömmliche Client/Sever-Beziehung eine zentrale Rolle spielen, sondern Software-Komponenten werden interagieren und mobile Agenten relativ frei in Datennetzen migrieren. Für Informationssysteme bedeutet das zum einen, daß die Daten, die sie Verarbeiten im Netzwerk verteilt liegen, und zum anderen, daß der Mechanismus, an Daten zu gelangen nicht ausschließlich RDA[5] ist, sondern daß vermehrt auch Agenten im Netz verteilte Datenquellen besuchen, um auf diese Weise nur die relevanten Daten einer Abfrage mit auf ihre Reise zu nehmen.

Offenheit Für die Interaktion von Programmen ist die Übereinkunft der Kommunikationsprotokolle unerläßlich. Davon sind nicht nur Netzwerkprotokolle, sondern auch alle anderen Schnittstellen betroffen. Um dies sicherzustellen gibt es „de facto“- und „de jure“-Normen (→[Nut92] Seite 8). Erstere sind alle diejenigen, die sich durch Zufall oder informelle Absprachen bilden, wie z.B. der IBM PC-AT Bus oder Internet-RFCs. Und letztere werden durch Kommittes, Organisationen und Vereine, wie die OMG, ISO etc., formuliert. Durch Offenheit gegenüber einer Norm oder einem Standard gewährleistet ein Produkt, daß es im Rahmen dieser Norm ohne zusätzliche Aufwendungen mit anderen Produkten, die dieselbe Norm respektieren interagieren oder kooperieren kann. Da Informationssysteme insbesondere nicht immer als monolitische Gebilde betrachtet werden können, ist gerade bei betriebswirtschaftlichen Informationssystemen, die häufig Daten aus externen Resourcen benötigen oder Daten in externe Systeme, z.B. Archivsysteme, auslagern, Offenheit ein entscheidendes Kriterium ihres Nutzens.

Systemkonstruktion in Tycoon

Durch das Tycoon-System mit der Programmiersprache *Tycoon Language*(Tl) (→[MS92, Mat93]) wurde eine Datenbankprogrammiersprache geschaffen, die wichtige Schlüsseleigenschaften (siehe Abbildung 1.3) in sich vereinigt, die die Modellierung und Systemkonstruktion erheblich vereinfachen.

Der Tycoon-Compiler steht in direkter Verbindung mit einem interaktiven Toplevel, der Eingaben einliest, compiliert und typüberprüft, um dann P-Code ähnlichen Byte-Code für eine virtuelle Stackmaschine direkt in den die Laufzeitumgebung repräsentierenden Objektspeicher einzufügen. So wird eine Interpretersystemen und 4GL-Sprachen ähnliche Interaktivität erreicht, ohne jedoch deren nachteilige Einschränkungen in der Typsicherheit oder ihrer mangelnden Orthogonalitätseigenschaften zu übernehmen.

Neben den genannten Eigenschaften zeichnet sich Tycoon durch einen sehr schmalen Sprachkern, orthogonales Typsystem und Funktionen sowie Threads als Werte erster Klasse aus.

Weiterhin skaliert Tycoon in verschiedenen Anwendungsdomänen gut, was durch ein flexibles Typsystem und einen hohen Grad an Orthogonalität und die Auslagerung aller Spracherweiterungen in externe Bibliotheken erreicht wird. Dadurch wird die Konstruktion von Systembibliotheken wie auch Applikationsbausteinen in einem einheitlichen Sprachrahmen unterstützt.

Eine weitreichende Offenheit wird durch die Integration externer Datenbank-, Information-Retrieval- und Präsentationsdienste erreicht. Diese Dienste sind derart in Tycoon integriert, daß der Umgang und die Programmierung mit ihnen typsicher innerhalb der Tycoon-Umgebung abläuft.

[5] Remote Database Access

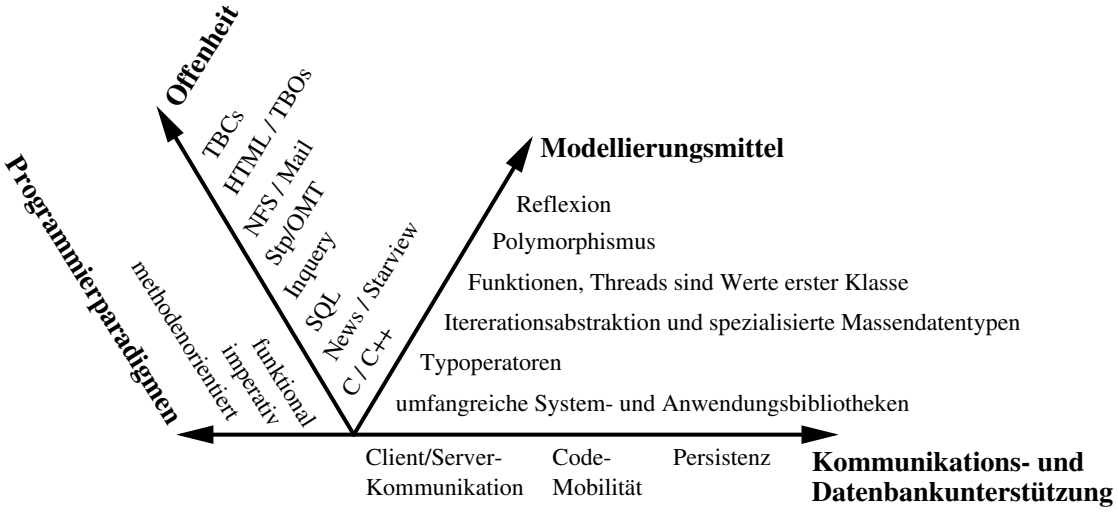


Abbildung 1.3: Orthogonale Tycoon-Systemeigenschaften und -Freiheitsgrade

Zusätzlich übersteigen die Kommunikationsmöglichkeiten des Systems die übliche Client/Server-Lösung, indem auch migrierende Objekte unterstützt werden (→[Mat96a]), so daß die Basistechnologie zur Entwicklung verteilter Agentensysteme bereits als Systemeigenschaft vorliegt.

Durch die genannten Eigenschaften ist die Tycoon-Sprachumgebung zur Konstruktion einer „systemverschmelzenden“ Sprachumgebung, wie sie schon in [MS93] und [Mat96a] vorgeschlagen wird geradezu prädestiniert.

1.1 Ziel der Arbeit

Damit stehen sich mit SAP R/3 und Tycoon zwei Systeme unterschiedlicher Leistungsfähigkeit und Zielsetzung gegenüber. SAP R/3 setzt seinen Schwerpunkt deutlich im betriebswirtschaftlichen Bereich und benutzt, historisch und wirtschaftlich bedingt, „primitive“ System- und Modellierungstechniken. Tycoon hingegen implementiert weitestgehend modernste Konzepte, besitzt aber noch keine betriebswirtschaftlichen Bibliotheken oder Anbindungen an entsprechende Systeme.

Ziel dieser Arbeit ist die Integration des SAP R/3-Systems in den sprachlichen Rahmen von Tycoon zu realisieren, so daß auf Basis der umfangreichen betrieblichen Datenbestände im R/3-System moderne Konzepte der Systemmodellierung und die Offenheit des Tycoon-Systems zur Verbesserung und Erweiterung der betriebswirtschaftlichen Leistungen des SAP R/3-Systems bzw. des Anwendungsfeldes der Tycoon-Programmierwerkbank erreicht wird.

1.2 Gliederung der Arbeit

Die Arbeit besteht neben dieser Einleitung aus den im folgenden beschriebenen Hauptkapiteln, einer Zusammenfassung der Ergebnisse mit einem Ausblick auf mögliche Nachfolgearbeiten und mehreren Anhängen:

Tl – eine Programmiersprache zur Systemintegration beschreibt die funktionale, strikt typisierte, polymorphe, persistente Programmiersprache Tycoon Language(Tl) unter dem Aspekt der Systemintegration.

Dazu wird eine grobe Beschreibung des Tycoon-Systems vorgenommen. Zur Modellierung stellt Tl Typen, Werte, Bindungen und Signaturen bereit, mit denen semantische Objekte der Arten Daten, Co-

de und Threads beschrieben werden können, welche ihrerseits von der Systemumgebung die räumliche Eigenschaft Mobilität und die zeitliche Eigenschaft Persistenz erben.

Die Basistechnologie zur Systemintegration in Tycoon bildet eine C-Funktionsaufrufschnittstelle, die mittels dynamischen Linkens zur Laufzeit Bindungen an externe Funktionen, die sich in Shared-Libraries befinden, herstellt. Außerdem wird durch die Indirektion von Tycoon-Funktionsabschlüssen auf Funktionsreferenzen die Erzeugung von automatischen Stubs für eine Callback-Funktionalität unterstützt. Damit können beliebige durch eine C-Aufrufschnittstelle verfügbaren Dienste auf Tycoon-Funktionen abgebildet werden.

Problematisch in diesem Zusammenhang sind die zentralen Eigenschaften Persistenz und Mobilität, die sich nur auf Objekte erstrecken, die in dem Tycoon-Objektspeicher abgelegt sind. Bei dem Binden externer Dienste werden jedoch häufig flüchtige Objekte erzeugt, die sich zwar innerhalb des Tycoon-Betriebssystemprozesses befinden, aber durch eine Prozeßunterbrechung oder eine Migration verloren gehen. Durch besondere Vorkehrungen für diese Objekte auf Basis eines Mechanismus für flüchtige Daten werden auch sie persistent und mobil.

Exemplarisch werden vier Gateway-Schnittstellen aus früheren Arbeiten auf ihre wichtigsten Eigenschaften hin untersucht und der Grad der Unterstützung für die räumliche und zeitliche Verteilung in Tycoon-Tl festgestellt.

Integrationsmöglichkeiten des SAP R/3 Systems gibt einen Überblick über die Systemarchitektur unter dem Aspekt der Integrierbarkeit in Fremdsysteme.

Als „3-tier“[6] Client/Server-Produkt untergliedert sich die Architektur in die Schichten der Präsentation, Applikation und Datenbank. Die Ebene der Applikationen besitzt ihrerseits eine eigene Architektur aus mehreren Prozessen.

Die wesentlichen informationstechnischen Aspekte des Systems R/3 sind seine Daten, Funktionen und Abläufe, die sich als SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen[7] herausstellen.

Auf dieser Grundlage werden die offenen Integrationsschnittstellen des SAP R/3-Systems untersucht, um die Funktionsbausteine, spezielle ABAP/4-Programme, als beste Interaktionsschnittstelle zum System zu identifizieren. Nachfolgend werden einige Grundlagen der Entwicklungsumgebung zur Entwicklung von ABAP/4-Programmen präsentiert und Möglichkeiten SAP-Tabellen zu modifizieren, ABAP/4-Programme zu erzeugen und auszuführen und SAP-Transaktionen zu starten, untersucht.

Die SAP R/3-Funktionsbibliothek Die Funktionsbausteine stellen die Basisfunktionalität der Client/Server-Architektur des Systems R/3. Sie sind die Service-Kommunikationsendpunkte der R/3-Middleware. Sie können sowohl lokal innerhalb eines Applikationsservers, als auch über Netzwerkschranken hinweg aufgerufen werden. Deshalb heißt die zugehörige Programmierschnittstelle *Remote Function Call*(RFC).

SAP bietet eine Programmierschnittstelle für externe Systeme an, die den RFC für die Verwendung in Programmen der Sprache C unterstützt. Damit bieten die Funktionsbausteine eine ideale Pforte für komplexe Datenbankänderungen und -Abfragen im System R/3.

Die Möglichkeiten und Einschränkungen des RFC, wie statischer Kontext, dynamischer Kontext, Aufrufsemantiken, Parametersignaturen und Ausnahmebehandlung geben den Rahmen Integrationsmöglichkeiten des SAP R/3 vor.

Konzepte zur Integration des SAP R/3 in Tycoon beschreibt die Anbindung des Systems SAP R/3 an die Programmiersprache Tycoon. Die Anforderungen an eine solche Schnittstelle ergeben sich klar aus der Motivation eine semantisch reichhaltige Integration der Möglichkeiten des SAP R/3 Systems in die Sprache Tl zu erhalten. Desweiteren sollen aber auch wichtige Eigenschaften der Tycoon-Programmierumgebung, wie Persistenz und Mobilität, berücksichtigt werden.

[6] Client/Server-Anwendung aus den Komponenten Client, Appplikationsserver, Datenbankserver (→[OHE96a])

[7] Es wird bewußt zwischen Transaktionen, mit denen üblicherweise Datenbanktransaktionen im Sinne des Read/Write-Modells[VGH93] gemeint sind, und SAP-Transaktionen unterschieden. In Literaturquellen zum System SAP R/3 werden häufig SAP-Transaktionen und Transaktionen synonym verwendet.

Wegen der unterschiedlichen Typsysteme und den damit verbundenen semantischen Abweichungen in den offenen Schnittstellen wird zunächst, basierend auf den Erkenntnissen der Untersuchung des SAP-Typsystems, die Abbildung primitiver Signaturen diskutiert, die eine Einbettung der ABAP/4 Basistypen in das Tycoon-Typsystem als Ergebnis hat.

Darauf aufbauend müssen, unter Berücksichtigung der unterschiedlichen Funktionsschnittstellen des RFC und der Tycoon-Funktionen, verschiedene Varianten der Bindungsarchitektur in Tycoon erörtert werden, von denen dann die präferierte und implementierte Variante der dynamischen Bindung mit polymorpher Aufruffunktion und dynamisch typisierten Parameterleisten beschrieben wird. Desweiteren werden die Verfahren zum Parametermarshalling und der Unterstützung von Persistenz und Mobilität vorgestellt.

Eine Erweiterung der polymorphen Funktionsschnittstelle bildet eine statisch typisierte Schicht, die essentiell kürzere Funktionsbindungen zuläßt, Typfehler schon zum Übersetzungszeitpunkt erkennt und das in der unterliegenden Schicht notwendige Hantieren mit automorphen Werten verbirgt. Zudem wird durch diese Schicht das Problem der mehrfachen Ergebnisparameter, die von Funktionen in Tycoon nicht direkt darstellbar sind, gelöst.

Zusammenfassung der Ergebnisse und Ausblick enthält noch einmal die wichtigsten Eigenschaften des Gateways in dem vorliegenden Entwicklungsstand. Desweitern werden mögliche Erweiterungen diskutiert und einige Nachfolgearbeiten, die zum Teil auf andere von SAP neuerdings etablierte Integrationsschnittstellen aufsetzen, vorgeschlagen.

1.3 Spezielle Notationen und graphische Modelle

An vielen Stellen dieser Arbeit werden zur Visualisierung der Zusammenhänge graphische Modelle angegeben. Um keine Unklarheiten in ihrer Semantik aufkommen zu lassen, werden die gebräuchlichsten Diagrammtypen hier zusammengefaßt erläutert. Man kann daraus ein intuitives Verständnis der Darstellungen entwickeln.

Die Bedeutung von Syntaxdigrammen kann in [BG84] Seite 156-162 nachgeschlagen werden. Sie sollten jedem Informatiker so geläufig sind, daß von einem allgemeinen Verständnis ausgegangen wird. Einige weitere Diagrammtypen sind hier ebenfalls nicht aufgeführt. Die betrifft dann entweder solche, deren Bedeutung im Text erläutert wird oder welche, denen eine Legende beigefügt ist.

Konkret beschreiben die hier aufgeführten Diagramme die Struktur, das Verhalten und die Berechnungen von Systemen. Sie bilden bestimmte Teilaspekte ab, ohne jedoch umfassend in jedem Detail zu sein. So daß eine Reduktion auf das Wesentliche erreicht wird.

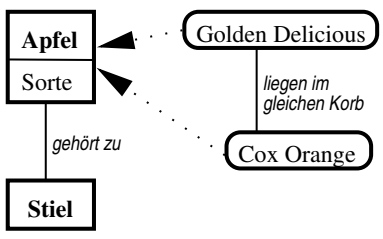
Die Systemstruktur läßt sich ausgezeichnet mit Klassen- und Objektdiagrammen aus [RBP+93] beschreiben. Sie bieten so reichhaltige Ausdrucksmöglichkeiten, daß an dieser Stelle nur ein Teil der dort aufgeführten Möglichkeiten Verwendung findet.

Eine Klasse wird durch einen rechteckigen Kasten symbolisiert. Die Klasse kann neben ihrem Namen Attribute oder Methoden enthalten.

Objekte, also Instanzen von Klassen, werden durch abgerundete Kästen dargestellt. Die Klasse liefert der Objektinstanz die notwendige Metainformation zur Interpretation.

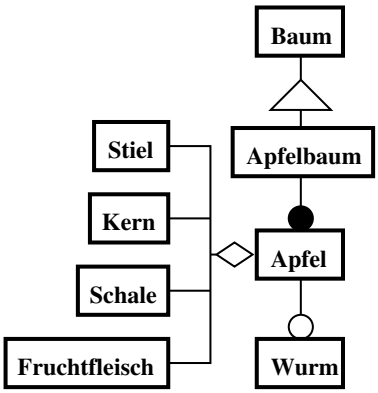
Ein gepunkteter Pfeil gibt eine Instanzenbeziehung von einem Objekt zu seiner Klasse an.

Assoziationen zwischen Klassen oder Objekten werden mit durchgezogenen Linien hergestellt, die mit ihrer Bedeutung beschriftet sein können.



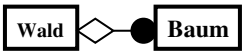
Für Klassen können die Assoziationen weiter qualifiziert werden. Dies hat direkte Auswirkung auf Assoziationen der Objektinstanzen. Insbesondere wird damit festgelegt welche Objekte in Beziehung

stehen und wieviele Beziehungen ein Objekt haben kann.



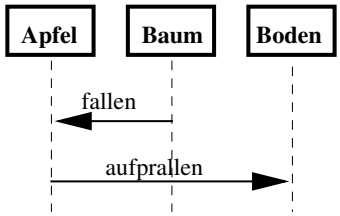
Ein Dach deutet eine Spezialisierung am flachen Ende und eine Generalisierung am spitzen Ende an. Auf diese Weise können Vererbungsbeziehungen im objektorientierten Sinn sehr gut dargestellt werden. Es werden den Eigenschaften der Klasse am generalisierten Ende des Daches zusätzliche spezialisierte Eigenschaften hizugefügt.
Der ausgefüllte Punkt gibt eine unbestimmte Multiplizität an, während der nicht ausgefüllte Punkt eine null oder eins Multiplizität bedeutet. So kann ein Objekt einer Klasse, die über eine Assoziation mit unausgefülltem Punkt zu einer anderen Klasse in Beziehung steht, mit nur höchstens einem Objekt der anderen Klasse in Beziehung stehen.

Das Karo drückt die Aggregation aller Klassen in der mit dem Karo direkt inzidierenden Klasse aus. Die Aggregation für sich beschreibt eine „Teil von"-Beziehung, im Unterschied zu der Generalisierung, die eine „vom Typ"-Assoziation angibt (siehe [RBP+93] Seite 72).
Auch Kombinationen der Piktogramme sind möglich. Zum Beispiel aggregiert ein Wald ein unbestimmte Anzahl von Bäumen, was sich, wie nebenstehend gezeigt, einfach visualisieren läßt.

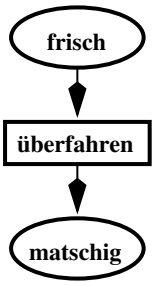


Das Verhalten von Systemen betrifft die Reihenfolge der Ereignisse, die in einem System auftreten können und die damit im Zusammenhang stehenden Zustandsänderungen. Man benötigt somit zweierlei Arten von Diagrammen, solche in denen die Ereignisse und ihre zeitliche Abfolge beschrieben werden, und andere, die Zustände und ereignisabhängig die Zustandsübergänge dokumentieren.

Als erstere werden Ereignispfaddiagramme benutzt, wie sie auch in [RBP+93] vorkommen. Es werden die betroffenen Objekte entweder links in einer Spalte oder oben in einer Zeile aufgelistet. Von jedem Objekt führt eine gestrichelte Linie weg. Zwischen diesen, die Objekte repräsentierenden Linien, markieren Pfeile Ereignisse vom auslösenden Objekt weg hin zum Empfängerobjekt. Die Ereignisse sind meistens mit ihrem Namen beschriftet. In einigen Ereignisflußdiagrammen ist die zeitliche Abfolge wichtig. Die Ereignisse treten dann in der Reihenfolge entsprechend von links nach rechts oder von oben nach unten ein.

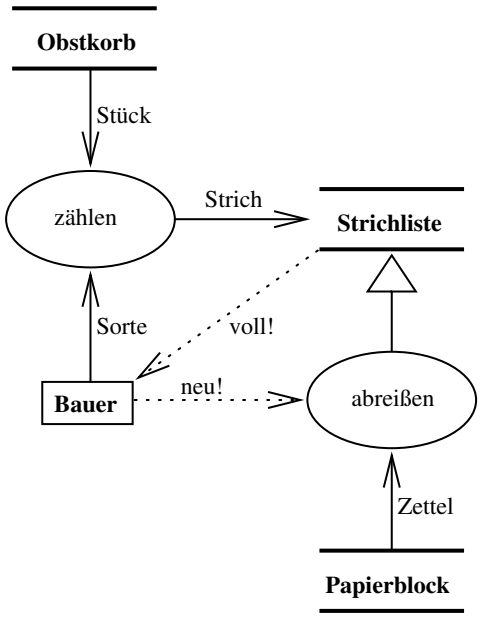


Die zur Darstellung der Systemzustände benutzten Diagrammtypen sind gewöhnliche Petrinetze (→[JV87]). Sie sind eine semantische Erweiterung des gängigen Modells für endliche Automaten in der Hinsicht, daß man auf den Zustandsübergängen Transitionen einführt, die Aktivitäten verrichten können. Formal ergibt sich daraus ein gerichteter bipartiter Graph, dessen Eckenmenge in die Stellen- und Transitionsmenge partitioniert sind. Kanten dürfen nur zwischen verschiedenartigen Ecken gezogen werden.



In dem in dieser Arbeit benutzten Zusammenhang werden die Stellen als Zustände interpretiert. Alle Attribute eines Objekts zu einem bestimmten Zeitpunkt bilden einen Objektzustand. Und alle Attribute aller Objekte eines Systems zu einem bestimmten Zeitpunkt bilden einen Systemzustand. Zustände werden durch Ellipsen gezeichnet.
Die durch rechtecktige Kästen dargestellten Transitionen bezeichnen Aktivitäten. Es können mehrere ausgehende Transitionen aus einem Zustand herausführen. Die Auswahl derjenigen, die im konkreten Anwendungsfall schaltet, wird durch die Benennung des Ereignisses, das eine Transition auslöst, getroffen. Diese Ereignisse sind auch Teil der Ereignisflußdiagramme.
Für jedes Zustands-Transitions-Diagramm wird angegeben, welches Objekt oder System es beschreibt. Das nebenstehende zum Beispiel beschreibt einen auf der Straße liegenden Apfel, der überfahren wird.

Die Berechnungen, die in Systemen stattfinden, hängen direkt mit den Daten zusammen, auf denen sie durchgeführt werden. Daher eignen sich hier Datenflußdiagramme, wie sie in [Raa91, RBP+93] Verwendung finden. Sie beinhalten Datenquellen und Senken, die gegebenenfalls in Strukturdiagrammen als Klassen wiederzufinden sind. Desweiteren erlauben sie aber die Darstellung der Datenflüsse zwischen Quellen, Senken, Speichern und Prozessen, sowie die dynamische Erzeugung von Speichern.



Quellen und Senken werden durch rechteckige Kästen symbolisiert. Sie repräsentieren Aktoren, die Daten produzieren oder konsumieren.

Prozesse werden als Ellipsen gezeichnet. Ein Prozeß transformiert eingehende Daten in ausgehende Daten. Dabei werden im Datenflußdiagramm zeitliche Präferenzen nicht spezifiziert. Vielmehr wird festgehalten welche Daten von wo kommen, wie die Datentransformation des Prozesses benannt wird und wo die Ergebnisdaten hinfließen. Die Datenflüsse werden als Pfeile mit durchgezogener Linie markiert. Eine Konsistenzbedingung (siehe [Fri91] Seite 99) dieses Diagrammtyps ist, daß Datenflüsse mit mindestens einem Prozeß verbunden sind.

Speicher werden als parallele Linien mit inliegendem Namen gezeichnet. Manchmal sind sie das Ergebnis eines Prozesses und werden dann als Objekt angelegt. Das wird als durchgezogener Pfeil mit hohler Pfeilspitze symbolisiert. Der häufigste Anwendungsfall ist aber der Speicherzugriff. Er wird durch einen durchgezogenen Pfeil mit schmaler Spitze gezeichnet. Die Pfeilrichtung zeigt die Richtung des Datenflußes an. Die Beschriftung bezeichnet das transportierte Datum.

Kapitel 2

Tl – eine Programmiersprache zur Systemintegration

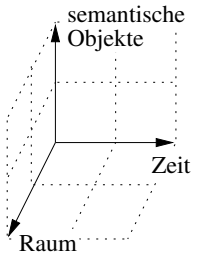
Als Datenbankprogrammiersprache vereinigt Tl im Tycoon-System Ansätze mit wichtigen Konzepten aus den beiden Bereichen der Datenbanken und der Programmiersprachen. Dies führt zu völlig neuen Methoden der Systemkonstruktion, die alternativ nur in heterogenen Umgebungen verschiedener Sprachen und Applikationsprogramme durchgeführt werden könnten.

Die Abstraktion der Eigenschaften Persistenz und Mobilität von programmiersprachlichen Objekten führt zu einem Modellierungsrahmen, der nahezu optimale Flexibilität bietet, um moderne Informationssysteme mit ihren Anforderungen bestehend aus QPRI und ESVO (siehe Abschnitt 1) zu realisieren. Durch Abbilden der Applikationssemantik auf die semantischen Basisprimitiven der Tycoon-Umgebung, Daten, Code und Threads, können auf einfache Weise neue Anwendungen geschaffen werden, die automatisch die zeitliche und räumliche Transparenz des Tycoon-Systems annehmen.

Eine weitere Methode Tycoon mit Applikationssemantik anzureichern besteht in der Gateway-Konstruktion. Unter der Annahme, daß bereits ein System oder ein Programm mit der gewünschten Applikationssemantik vorliegt und offene Interaktionsschnittstellen besitzt, wird in Tycoon eine Bibliothek geschaffen, die durch proprietäre Mechanismen auf diese Interaktionsschnittstelle zugreift und deren Dienste durch einen oder mehrere abstrakte Datentypen nach Tycoon importiert. Auf diese Weise können in Tycoon neue Anwendungen unter Benutzung dieser externen Dienste entwickelt werden, ohne daß diese Applikationen direkt in Tycoon programmiert werden müssen.

Dieses Kapitel beschreibt die wesentlichen Grundlagen der Tycoon-Systemumgebung mit einem Schwerpunkt auf den Eigenschaften, die wesentlich sind für den Bau moderner Informationssysteme und die Gateway-Konstruktion. Insbesondere sind das in Abschnitt 2.1 die programmiersprachlichen Basisobjekte, sowie die Abstraktionsmechanismen für deren räumliche und zeitliche Verteilung. In Abschnitt 2.2 werden die offenen Schnittstellen zur Systemintegration beschrieben. Schließlich folgt eine Untersuchung exemplarisch ausgewählter Gateway-Integrationen mit der Zusammenfassung der wichtigsten Integrationseigenschaften in Abschnitt 2.3.

2.1 Eigenschaften der Sprache Tl und ihre semantischen Objekte



[Mat96a] beschreibt die Tycoon-Welt als kombinatorischen Raum aus semantischen Objekten, Persistenz und Mobilität. Dieser kombinatorische Raum wird in einer Laufzeitumgebung gehalten, die wiederum in einem persistenten Objektspeicher abgelegt ist. Das Tycoon-System[1] in der für diese Arbeit verwendeten Fassung besteht aus einem Compiler, der Programmtext der Sprache Tl übersetzt und typüberprüft, einer virtuellen Maschine(TVM), die die Interpretation der als Bytecode abgelegten Objekte zur Laufzeit vornimmt, und einer Objektspeicherschnittstelle(TSP), die Transparenz gegenüber dem konkret verwendeten Objektspeicher bietet.

Die Granularität der Zusammenfassung programmiersprachlicher Konstrukte ist das Modul für kleinere Aufgaben, die sich als abstrakter oder semiabstrakter Datentyp realisieren lassen. Für die Programmierung umfangreicher Applikationen *im Großen* stellt Tycoon-Tl ein Bibliotheks-Konzept (→[Koc96, Mat93]) zur Verfügung, das eine Strukturierung durch Vereinigung von Modulen und Bibliotheken in Bibliotheken erlaubt. Bibliotheken werden extern übersetzt und dann in die Laufzeitumgebung importiert. Wahlweise können Befehle aber auch interaktiv über den Toplevel in die aktuelle Laufzeitumgebung eingegeben werden. Eine Beschreibung der Benutzung der interaktiven Tycoon-Systemumgebung befindet sich in [MMM93].

Die Einführung in die Programmierung mit der Sprache Tycoon Language findet sich in [MMS94]. In [MS92] wird eine Definition durch die Beschreibungen der Grammatik, einer abstrakten Syntax und der statischen Semantik gegeben. Die Syntax basiert auf einer LL(1)-Grammatik (→[FRJL88]), in der jedes sprachliche Konstrukt durch ein identifizierendes Schlüsselwort eingeleitet wird und wichtige Schachtelungsprimitiven durch ein `end` abgeschlossen werden.

Das semantische Modell von Tl basiert auf Typtheorie höherer Ordnung (→[BL84, CMMS91]). Ihre wichtigsten Elemente sind Signaturen, Bindungen, Typen, und Werte:

Signaturen Signaturen sind Kombinationen aus Namen und Typen. Sie bilden partielle Spezifikationen in Modul-Schnittstellen, Typbindungen oder Parameterlisten von Funktionen. Durch Signaturen wird ihre Struktur definiert, so daß ihre Namen und Typen modulo Polymorphismus festgelegt sind:

```
sum :Fun(i, j :Int) :Int                    (* eine Funktionssignatur *)
Budget :Tuple zweck :String  betrag :Int end   (* eine Tupel-Signatur *)
```

Bindungen Durch Bindungen werden Tl-Objekte mit Namen assoziert. Über solche Namen können diese Objekte später referenziert werden. Dadurch findet zum einen eine Abstraktion von konkreten Objekten statt und zum anderen wird ein imperativer Programmierstil ermöglicht.

Nach [BL84] sind Bindungen vier-Tupel aus Namen, Wert, Typ und einem Bit, das angibt, ob die Bindung veränderlich oder konstant ist. Es werden Wertbindungen `let i = 3` und Typbindungen `Let MyType =Int` unterschieden. Wenn ein Name wiederholt gebunden wird, überschreibt diese neue Bindung eine bestehenden Bindung an den gleichen Namen.

Der Sichtbarkeitsbereich von Namen beschränkt sich in Tl auf die die Bindung einschließende Blockanweisung. Dabei erbt ein Block alle Bindungen seiner direkten Umgebung:

```
let i = 3        (* i wird an 3 gebunden *)
  begin
  let i = 5      (* i wird an 5 gebunden *)
  end
i                (* i evaluiert zu 3 *)
```

Typen Das Typsystem von Tl ist in der Art eines vollständigen mathematischen Verbands (→[KK71]) organisiert. Die zugehörige Ordnungsrelation[2] ist die Subtypbeziehung <:. Das Einselement oder der nichterfüllbare Typ ist `Nok` und das Nullelement bzw. die triviale Typspezifikation ist `Ok`. Dementsprechend sind alle vorkommenden Typen Subtypen von `Ok` und Supertypen von `Nok`. Das gesamte Typsystem ist als Termersetzungssystem über induktiv definierte Subtypbeziehungen (→[MS92]) beschrieben.

Die weitere Struktur des Typsystems untergliedert sich in Basistypen, homogen und heterogen strukturierte Typkonstruktoren und Typoperatoren, die rekursiv und höherer Ordnung sein dürfen. Subtypbeziehungen sind ausschließlich struktureller Natur mit Ausnahme der Typen `Nok` und `Ok`.

Als atomare Basistypen stehen `Bool`, `String`, `Char`, `Int` und `Real` zur Verfügung. Sie stehen paarweise in keiner gegenseitig Subtypbeziehung.

Eine kompliziertere Struktur ergibt sich durch die Typkonstruktoren. Die heterogenen Typkonstruktoren `Tuple` und `Record` aggregieren jeweils mehrere Signaturen, die beim `Tuple` auch anonym sein dürfen. Der `Tuple`-Konstruktor aggregiert die Signaturen unter Bewahrung der Reihenfolge, während beim `Record`-Konstruktor die Identifikation der Subsignaturen über ihre Namen vorgenommen wird. Die Subtypbeziehung wird sowohl bei `Tuple`s, wie auch bei `Record`s auf die Subsignaturbeziehung der eingetragenen Signaturen zurückgeführt.

Der Typkonstruktor `Fun` für Funktionen besitzt mit der Kontravarianzregel die komplizierteste Subtypbeziehung. Für zwei Funktionen müssen die Anzahl der Parametersignaturen übereinstimmen, die Argumentparameter der Subtyp-Funktion müssen Supertypen derjenigen der Supertyp-Funktion sein und der Ergebnisparameter der Subtyp-funktion muß ein Subtyp desjenigen der Supertyp-Funktion sein. Die Argumentparameter der Subtyp-Funktion verhalten sich also kontravariant im Gegensatz zu dem kovarianten Ergebnisparameter.

Der homogene Typkonstruktor `Array` erlaubt die Spezifikation eines Feldes variabler Einträge, fester Größe und gleichen Typs. Zwei `Array`s stehen genau dann in Subtypbeziehung, wenn sie über Typen erklärt sind, die ihrerseits in Subtypbeziehung stehen.

Typoperatoren abstrahieren Typ-Bindungen. Ähnlich einer Funktion haben sie Parameter mit einer Typ-Spezifikation und ergeben immer eine Mindest-Spezifikation. Allerdings werden Typoperatoren im Rahmen der statischen Typprüfung immer zum Übersetzungszeitpunkt evaluiert. Die Subtypbeziehung zwischen Typoperatoren ergibt sich analog der Kontravarianz-Regel für Funktionen.

Let Konto(Waehrung <:**Int**) <:**Ok** = **Tuple** name :**String** ktoStand :Waehrung **end**

Eine Erweiterung der Typoperatoren bilden die rekursiven Typenoperatoren. Sie erlauben die Konstruktion beliebig großer Werte homogenen Typs.

```
Let Rec List(T <:Ok) <:Ok = Tuple
                              case node with val :T  next :List(T)
                              case nil
                            end
Let KontenGruppe = List(Konto)
```

Dabei sind auch Typoperatoren höherer Ordnung zugelassen:

[2] reflexiv, transitiv und identitiv

```
Let KontenPlan( Par <:Oper() Int ) = Par()
```

Letztendlich sind alle Typen orthogonal kombinierbar, so daß ein reichhaltiges, leicht skalierbares Typsystem höherer Ordnung vorliegt. Dieses wird durch den im Zusammenhang mit Werten erklärten Polymorphismus in den drei Varianten des Subtyp-, Parametrischen und Reflektiven Polymorphismus weiter flexibilisiert.

Werte In Tycoon sind Daten Werte als konkrete Ausprägungen verschiedener Typen. Beispiele für die Werte sind:

```
true false                          (* Werte vom Typ  at10.0ptBool *)
1                                   (* Wert vom Typ Int *)
301.41                              (* Wert vom Typ Real *)
"Budget"                            (* Wert vom Typ String *)
array 1 2 3 end                     (* Wert vom Typ Array(Int) *)
tuple "Kosten" "Kostenstelle"  let betrag=3500 end
                        (* Wert vom Typ Tuple :String :String  betrag :Int end *)
tuple_case node of List(Int) with 33 tuple_case nil of List(Int) end end
                        (* Wert vom Typ List(Int) *)
```

Eine Sonderrolle nehmen Funktionen ein. Sie stellen nicht nur einen Wert erster Klasse dar, sondern sind in Kombination mit der Funktionsapplikation `()` auch das Basiskonstrukt für späte Bindung. Denn eine Funktion kann durch Applikation ihren Rumpf ausführen und dann einen neuen Wert ergeben. Dementsprechend Verhalten sich die Funktionsabstraktion eines Wertes durch Voranstellen von `fun()` und die Applikation durch Nachstellen[3] von `()` komplementär.

Jede Funktion hat eine eventuell leere Parameterliste. Sie kann Typ- und Wertsignaturen aufnehmen. Eine entsprechenden Funktionsapplikation muß die passende Typen und Werte enthalten, damit sie zur Evaluation der Funktion als Aktualparameter an die Parametersignaturen gebunden werden können:

```
let gewinn = fun(umsatz, kosten :Real) :Real umsatz-kosten
  (* Wert vom Typ :Fun( :Real :Real ) :Real *)

gewinn(33000.00 29800.00)
  (* berechnet den Gewinn eines Geschaefts mit 33000.00 Umsatz
     und 29800.00 Kosten als 3200.00 *)
```

Die Parameterbindungen sind polymorph, so daß Funktionen mit Werten unterschiedlichen Typs aufgerufen werden können:

Subtyppolymorphismus Als Aktualparameter einer Funktion können Werte einer spezielleren Spezifikation entsprechend der Subtypbeziehung <:, als die geforderte angegeben werden. Der Subtyppolymorphmismus garantiert existentiell die geforderte Spezifikation der Signatur:

Parametrischer Polymorphismus Durch die Angabe von Typ-Parametern in Funktionssignaturen werden typunabhängige Berechnungen formuliert. Typ-Parameter können durch die Angabe einer oberen Schranke beschränkt werden, bieten aber fuer Subtypen universelle Quantifizierung. Die Verwendung von Typparameter in auf sie folgenden Wertparametersignaturen ist zulässig.

[3] `fun()a()` ist in Tl nicht korrekt, weil nicht kontextfrei entschieden werden kann, ob das zweite Klammerpaar die Applikation von `a` oder `fun()` durchführen soll!

Reflektiver Polymorphismus Der abstrakte Datentyp `typeRep.T`, dessen Werte Typbeschreibungen sind, erlaubt typgesteuerte Berechnungen mit typabhängigen Code-Fragmenten (→[Gei95, Koe96]). Er stellt Funktionen zum Generieren und zum Inspizieren von Typbeschreibungen zur Verfügung.

2.1.1 Semantische Objekte

Neben der Partitionierung der Tycoon-Semantik in Typen, Werte, Bindungen und Signaturen, die Basiskonzepte auf sprachlicher Ebene sind, ist es sinnvoll den Raum der Tycoon-Werte in seine wichtigsten drei Klassen von Daten, Code und Threads (→[MS94a]) zu zerlegen. Das Kriterium, um Werte derart zu unterscheiden liegt in ihrer Evaluationssemantik:

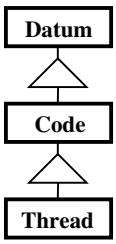
Daten Als Daten werden Tl-Werte erster Klasse bezeichnet. Hierunter fallen alle Tycoon-Objekte, die keine Typen sind, also atomare Werte, strukturierte Werte, Funktionen und auch Threads. Die Evaluation von Daten ergibt wieder ihren orginalen Wert.

Code Als Code werden in Tycoon alle Funktionen bezeichnet. Funktionen kapseln Berechnungsvorschriften in einer Lambda-Ausdrücken ähnlichen Form[4]. Insbesondere haben sie also eine Formalparameterliste, einen Ergebnistyp und einen Funktionsrumpf. Bei einer Funktionsapplikation (siehe Seite 14) evaluieren sie durch Auswerten des Funktionsrumpfes, indem die Aktualparameterwerte an die Formalparameter der Funktion gebunden werden und anschließend in dieser neuen Umgebung ihr Rumpf ausgewertet wird.

{**fun**(x, y :**Int**) x+y}(1 4) *(* evaluiert zu 5 *)*

Threads Als Erweiterung von Code um Zustandssemantik sind Threads, sogenannte „leichtgewichtige " Prozesse (→[CDK94, Tan95]), die eine Systemabstraktion für Aktivitäten darstellen. Dabei handelt es sich um nebenläufig ausgeführte Funktionen desselben Prozesses, deren Berechnungsablauf explizit im gesamten Sichbarkeitsbereich des Threads, also auch reflektiv vom Thread selbst, gesteuert werden kann. Ein Tycoon-Thread kann sich in den Zuständen *blocking*, *suspended*, *running*, *terminated* und *exception* befinden, der entweder von ihm selbst oder von einem anderen Thread desselben Tycoon-Prozesses beeinflußt werden kann.

Interessant ist ein qualitativer Zusammenhang zwischen Daten, Code und Threads. Daten sind Objekte erster Klasse der Sprache Tl. Code ist ein durch den Funktionsoperator `Fun` abstrahiertes Datum und selbst wieder Objekt erster Klasse. Ein Thread ist Code mit zusätzlicher Ausführungssemantik und ebenfalls Objekt erster Klasse. In [MS94b, MS94a] wird diese Eigenschaft, Objekt erster Klasse zu sein, durch ihre Bindefähigkeit beschrieben. Die semantischen Objekte lassen sich untereinander orthogonal binden, so daß Daten Code und Code Threads binden können.
Trotzdem stehen diese semantischen Objekte wie nebenstehend dargestellt in der beschriebenen Spezialisierungsbeziehung zueinander.



2.1.2 Persistenz

Alle Tycoon-Objekte, Daten, Code und Threads, sind durch die Konzeption der Systemumgebumg bereits persistent. Die Persistenz wird durch einen Mechanismus für flache Transaktionen auf der Ebene des *Tycoon Store Protocol*(TSP →[MMS95]), einer untypisierten persistenten Speicherhalde, erreicht. Dazu kann durch die Operation `store.commit()` ein Fortschreiben des aktuellen Systemzutands, bzw. durch `store.rollback()` das Zurücksetzen auf den Zustand nach dem letzen Commit durchgeführt werden. Der entscheidende Unterschied zu herkömmlichen Datenbanksystemen ist die Orthogonalität

[4] In der Tat entsprechen Tycoon-Funktionen einem typisierten Lambda-Kalkül zweiter Ordnung.

dieses Ansatzes, der nicht zwischen Datenbankrelationen und Programmvariablen unterscheidet und deshalb als *Persistenzabstraktion* bezeichnet wird.

Eine konzeptuelle Erweiterung dieses Mechanismus flacher Transaktionen wird in [Nie92] eingeführt. Insbesondere werden Transaktionen, Fehlererholung und Integritätskontrolle für die Anwendungsprogrammierung untersucht. Unter Ausnutzung Funktionen höherer Ordnung und erster Klasse, wird das Modul `transaction` der Bibliothek `dbenv` im „add on"-Ansatz (→[Mat93]) implementiert. Es erlaubt neben flachen Transaktionen mit einer Undo-Log gesteuerten Fehlererholung auch das Setzen von Sicherungspunkten und die Verschachtelung von Transaktionen zur feinkörnigeren Auflösung sehr umfangreicher Transaktionen.

Die beschriebenen Mechanismen zur Persistenzabstraktion und Transaktionensteuerung sind eine wichtige Grundlage, um langlebige, sichere Applikationen zu entwickeln. Ein wesentlicher Aspekt der Tycoon-Systemumgebung ist die Integration externer Systeme (vgl. Abschnitt 2.3), die die Berücksichtigung externer Daten, die nicht der Tycoon-Persistenz unterliegen, notwendig macht. [Kor95] nimmt eine systematische Analyse der Art der Daten vor und unterscheidet nach den drei Kriterien Herkunft, Transaktionalität und Persistenz. Die Herkunft kann entweder Tycoon-intern oder -extern sein. Wenn es sich um externe Daten handelt, sind sie eventuell durch externe Mechanismen bereits transaktional abgesichert. Wenn sie extern und nicht transaktional abgesichert sind, bleibt immer noch die Möglichkeit, daß sie einer externen Persistenz unterliegen, die für ihre Dauerhaftigkeit sorgt. Für die Systemintegration sind die internen und die flüchtigen externen Daten von Bedeutung.

Für die Behandlung der Tycoon-internen Daten wird das oben angeführte Konzept der Sicherungspunkte in [Kor95] auf einer auf das TSP aufsetzenden Schicht zu persistenten Sicherungspunkten (→[GR93]) ergänzt. Neben der Persistenz der Sicherungspunkte liegt ein wesentlicher Unterschied im Fehlererholungsalgorithmus, der sich auf im Gegensatz zu kompensierenden Operationen aus [Nie92] auf die Erzeugung von Versionen verläßt.

Behandlung flüchtiger Objekte

Flüchtige externe Daten, die im Zusammenhang mit Tycoon-externen Betriebsmitteln oder Gateways als flüchtige Prozeßspeicherobjekte des Tycoon-Systems erzeugt werden, unterscheiden sich grundsätzlich dahingehend, daß sie selbst nicht der Persistenzabstraktion unterliegen und damit dem Tycoon-System unzugänglich bleiben. Es sind jedoch die Operationen ihrer Erzeugung bekannt. Daher wird jede Operation, die externe Daten erzeugt, zusammen mit einer kompensierenden Operation in einem Operations-Log erfaßt (→[Kor95]). Beim Zurücksetzen[5] der laufenden Transaktion wird diese Historie der kompensierenden Operationen rückwärts aufgerufen und bei einem Wiederanlaufen[6] werden die Operationen des Operations-Logs noch einmal durchgeführt.

Dieser Mechanismus wird in [Mat96a] als *Pseudo-Persistenz* bezeichnet und durch das Modul persistent mit folgender vereinfachten Funktion zur Erzeugung pseudopersistenter Werte implementiert:

```
persistent.create(V <:Ok  create() :V  destruct(:V) :Ok) :T(V)
```

2.1.3 Mobilität

Die semantischen Objekte des Tycoon-Systems(Daten, Code und Threads), unterliegen einer orthogonalen Mobilität. Im Vergleich zur konventionellen Client/Server-Kommunikation (→[CDK94]) verteilter Systeme, in der ausschließlich Datenwerte als Parameter eines entfernten Prozedur- oder Funktionsaufrufs ihren Ort ändern, verlagert sich ein migrierendes Objekt komplett an einen neuen Ort. Diese Mobilität der Tycoon-Objekte wird durch ein transitives Ausschneiden der referentiellen Hülle des aktiven Threads aus dem Objektgraphen, abgesehen von Sonderbehandlungen lokaler und ubiquitärer Objekte (→[Pou96]), mit anschließender Erzeugung der externen Repräsentation TXR[7] rea-

[5] Bei Abbruch der Transaktion; siehe R3-Recovery in [Reu87]
[6] Typisch nach einem Systemausfall; siehe R2-Recovery in [Reu87]
[7] Tycoon eXternal Reprasentation

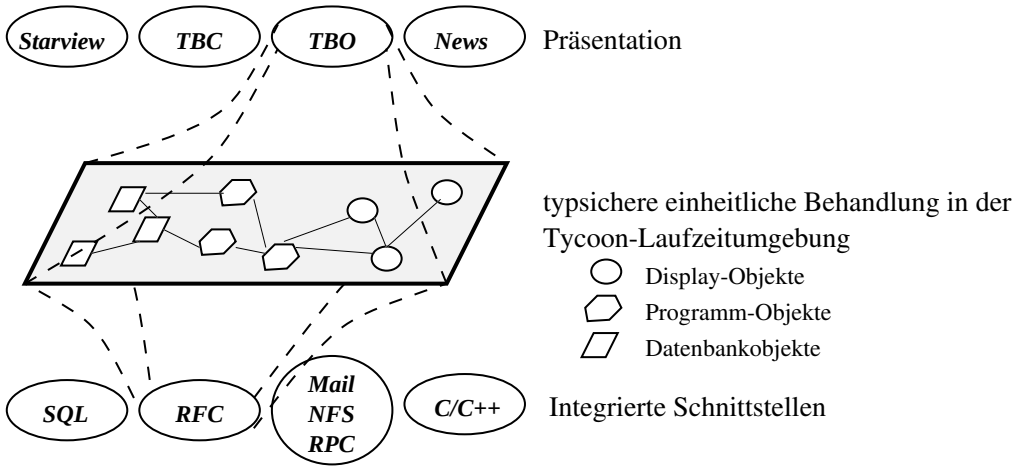


Abbildung 2.1: Tycoon erlaubt eine einheitliche Behandlung aller Objekttypen

lisiert (→[Mat96a]). Durch eine einfaches Byte-weises Übertragen und Einlesen in einen anderen Objektspeicher wird der ursprüngliche Objektgraph wieder hergestellt.

Behandlung flüchtiger Objekte

Wie bei der Persistenzabstraktion nehmen externe Daten auch bei der Mobilität eine Sonderrolle ein, da sie ebenfalls nicht von dem Algorithmus zur Linearisierung von Teilstrukturen des persistenten Objektspeichers in das Transportformat TXR erfaßt werden. [Mat96a] behebt diesen Mangel durch Einführen eines Operations-Log für die Objektmigration. Diese als *Pseudo-Mobilität* bezeichnete Strategie setzt jedoch die ubiquitäre Existenz aller beteiligten externen Betriebsmittel vorraus.

Nachdem in den vorangegangenen Abschnitten eine an Modellierung und zentralen Eigenschaften orientierte Darstellung des Tycoon-Systems mit der Programmiersprache Tl geliefert wurde behandeln die folgenden Abschnitte den Anwendungsfall der Integration von Fremdsystemen in die Tycoon-Umgebung.

2.2 Integration externer Systeme

Im Vergleich zu üblichen 3GL-Sprachen, die optimale Skalierbarkeit aber eine Trennung zwischen Datenbank-, Applikations- und Displayobjekten vornehmen, und 4GL-Sprachen, die zwar volle Integration aller beteiligten Objekttypen bietet, jedoch nur extrem schlecht skalieren, folgt Tycoon einem Ansatz, der durch Generalisierung und Orthogonalität die Vorteile von 3GL- und 4GL-Sprachen in sich vereinigt (→[MS93]).

Tycoon integriert Datenbank-, Applikations- und Displayobjekten in die typsichere, polymorphe Sprache Tl. Die Menge der Applikationsdienste zergliedert sich in interne Bibliotheken für Massendatentypen, Iterationsabstraktion etc. und Integrationsbibliotheken, die externe Applikationsdienste über Gateway-Konstruktionen auf Objekte der Programmiersprache Tl abbilden. Wie in 2.1 abgebildet sind das Präsentationsdienste(Starview, TBC, TBO, News), Anwendungsdienste(SQL, Mail, ...) und Systemdienste(RPC, NFS, C/C++). Die so integrierten Dienste profitieren automatisch von den Eigenschaften des Tycoon-System, wie Persistenz, Mobilität und Typsicherheit. Der Anwendungsentwickler konzentriert sich nun auf die Modellierung in Tl und ist von der Anpassung verschiedener Darstellungsformen in 3GLs bzw. der Überbrückung sprachlicher Ausdrucksschwächen in 4GLs entbunden.

Im folgenden werden die wichtigsten Integrationsschnittstellen des Tycoon-Systems vorgestellt.

Dazu gehören einerseits diejenigen der Client/Server-Kommunikation und andererseits die C-Call-Schnittstelle zum Einbinden externer Bibliotheken.

2.2.1 Client/Server-Kommunikation

Grundlage der Client/Server-Kommunikation in Tycoon sind Sockets und RPCs. Sie spielen aber eine untergeordnete Rolle in Tycoon. Zwar stellen sie die Basismechanismen für die Mobilität zur Verfügung, jedoch verlangt die Programmierung von Mobilität, die als Abstraktion[8] primitiver Client/Server-Mechanismen gesehen werden kann, keine Kenntnisse dieser Protokolle, deckt aber bereits deren Möglichkeiten ab.

Socketkommunikation

Als Basis einer Implementation für entfernte Prozeduraufrufe wurde in [Göl96] die UNIX-Socket-Schnittstelle [CDK94, Ste92] als abstrakter Datentyp implementiert. Mit Funktionen zur Erzeugung von Sockets, dem Binden an lokale Namensräume, dem Binden zwischen Kommunikationspartnern, dem Versenden und Empfangen von Daten bietet die Socket-Schnittstelle einen einfachen Mechanismus für abstraktere Protokolle, wie etwa die nachfolgen beschiebene RPC-Kommunikation.

Entfernte Prozeduraufrufe

Einen Ansatz zur RPC-Kommunikation behandelt [Joh95]. Auf Basis der RPCs von Sun oder DCE wird die verteilte Client/Server-Programierung mit den Teilaufgaben Dienstidentifikation, Dienstbindung, Marshalling, Datentransfer, Dispatching unterstützt. Die Dienstspezifikation findet dabei durch die Angabe einer Modulschnittstelle vollständig auf der Basis der Sprache Tl statt. Dies hat den Vorteil alle Möglichkeiten der Sprache, wie Polymorphismus, Typoperatoren etc. zu unterstützen. Der zugrundeliegende RPC dient lediglich dem verschicken von opaquen Byte-Strömen im TXR-Format (vgl. Abschnitt 2.1.3). Durch die Verwendung dynamischer Typen für die Schnittstellenspezifikation und des Subtyptests auf selbigen kommt diese Implementation ohne die aufwendige Erzeugung statischer Stubs aus.

Die Erkenntnis, daß in Sprachen mit dynamischer Typisierung, Funktionen höherer Ordnung und einem einheitlichen externen Speicherformat, ein einfacher datenstromorientierter Übertragungsmechanismus zur Realisierung einer Client/Server-Architektur ausreicht, wird in [Göl96, Mat96a] dazu genutzt eine leistungsfähige Infrastruktur aufzubauen. Auf der oben genannten Socket-Schnittstelle aufsetzend, wird ähnlich wie im zuvor beschriebenen Ansatz eine Client/Server-Kommunikation für entfernte Funktionsaufrufe implementiert. Zusätzlich werden jedoch umfangreiche Mechanismen zur Abstraktion von konkreten Netzwerkadressen[9] zu einem logischen Adressierungsschema, sogenannten Service-Ids, und die zugehörige Infrastruktur für die Rückabbildung der logischen auf physikalische Adressen entwickelt.

2.2.2 Externe Bibliotheken

In die Programmierumgebung Tyccon-Tl können externe dynamisch bindende Bibliotheken der Programmiersprache C bidirektional eingebunden werden (→[MMM93]). Da viele Applikationssysteme, insbesondere auch SAP R/3, Programmierschnittstellen für C-Programme bereitstellen, kann damit die Integration dieser Systeme in das Tycoon-System realisiert werden. Die Basismechanismen dieser Bindungen sind die im folgenden vorgestellten C-Call- und C-Callback-Schnittstellen für den Aufruf von C-Funktion aus Tl und umgekehrt.

[8] Diese Abstraktion ist vergleichbar mit der, die Isolationsmechanismen von Transaktionen gegenüber expliziter Synchronisation bieten.

[9] Eine Netzwerkadresse für einen RPC-Dienst besteht üblicherweise aus einer Rechneradresse, z.B. der vierstelligen Internetadresse 134.10.90.55 und einer prozeßidentifizierenden Port-Nummer

C-Calls

Der Aufruf externer C-Bibliotheken wird durch die vordefinierte Funktion

bind(Function <:**Ok** library, label, format :**String**) :Function

realisiert. Dabei beschreibt `Function` den Typ der Funktion, `library` ist eine vollständige Pfadangabe auf die Bibliothek, `label` enthält den Namen der Funktion und `format` beschreibt die Parametertypen, die entsprechend den Konventionen der Programmiersprache C nur atomar sein dürfen. Der durch `Function` vorgegebene Funktionstyp impliziert bereits die Parametertypen, die als Steuerzeichen (→[MMM93]) zu Marshalling-Zwecken noch einmal in `format` übergeben werden.

Das Aufsuchen und die Bindung an eine derart deklarierte C-Funktion findet dynamisch beim Aufruf statt, wodurch eine Entkopplung von statischen Abhängigkeiten diese Deklaration persistent und mobil[10] macht.

C-Callbacks

Durch das Modul `cCallback` wird die Generierung primitiver C-Funktionsreferenzen aus Tl-Funktionen höherer Ordnung unterstützt:

cCallback.new(Function <:**Ok** function :Function format :**String**) :T(Function)

Der Parameter `Function` muß einen Funktionstypen erhalten, `function` ist die passende Funktion dazu und `format` beschreibt die Parameterliste dieser Funktion durch Steuerzeichen.

Eine so generierte Funktionsreferenz kann als Parameter einem C-Call mit übergeben werden. Die aufgerufene C-Funktion kann damit ihrerseits eine Tl-Funktion zurückrufen.

Abbildung strukturierter Parameterwerte

Diese C-Call/Callback-Schnittstelle unterstützt ausschließlich atomare Werte für Parameterpositionen. Viele Programmierschnittstellen in der Sprache C erfordern jedoch die Übergabe von Referenzen auf komplette Strukturen an ihre Funktionen. Es gibt drei Möglichkeiten diesen „Mismatch“ zu umgehen:

Zugriffsfunktionen Es wird eine Funktionsschicht zwischen Tl-Programm und Applikations-API eingefügt, die einen entsprechenden Speicherbereich verwaltet. Desweiteren befinden sich Zugriffsfunktionen zum Lesen und Schreiben einzelner Slots dieses Speicherbereichs, sowie eine Funktion zum Aufruf der API-Funktion mit der Struktur als Parameter in dieser Schicht. Diese Zwischenschichtfunktionen werden ihrerseits im Tl-Programm gebunden.

Direkter Speicherzugriff Über das Modul `word` können Prozeßspeicherbereiche direkt inspiziert und modifiziert werden, so daß innerhalb des Tl-Programm bei Kenntnis des Speicherlayouts auf der C-Halde eine entsprechende Struktur aufgebaut werden kann.

Objektspeicherzugriff Ähnlich wie bei Zugriffsfunktionen wird eine Funktionsschicht zwischen das Tl-Programm und die Applikations-API eingefügt. Jedoch enthält diese nur eine Funktion, die über Aufrufe von TSP-Funktionen direkt Tl-Objekte aus dem Objektspeicher ausliest. Diese Variante setzt voraus, daß der Programmierer genaue Kenntnisse der TSP-Schnittstelle und des Objektspeicherformats hat.

Das umgekehrte Problem, aus einem auf der C-Halde vorliegendem strukturiertem Datum ein Tl-Tupel zu erzeugen, kann ebenfalls entweder über Zugriffsfunktionen, direkten Speicherzugriff oder einen Objektspeicherzugriff gelöst werden. Problematische Datentypen, wie nicht null-terminierte Strings, die vom Marshalling-Algorithmus des Tycoon-Systems nicht korrekt behandelt werden, bedingen eine der letzeren beiden Methoden des direkten Speicherzugriffs oder des Objektspeicherzugriffs.

[10] Die Mobilität setzt die ubiquitäre Existenz der Bibliothek vorraus.

Alle hier aufgeführten Methoden zur Behandlung nicht atomarer Parameterwerte haben das Problem flüchtige Werte auf der C-Halde zu erzeugen mit den entsprechenden Konsequenzen für Persistenz und Mobilität (vgl. Abschnitte 2.1.2, 2.1.3).

Extern ausgelöste Ausnahmen

Bei einer Anbindung externer C-Bibliotheken nach der Zugriffsfunktionen-Methode, die eigene Algorithmen für Parameterkonvertierungen und spezielle Protokolle enthält, können Ausnahmesituationen eintreten. Eine häufige Form der Ausnahme-Indikation der Programmiersprache C, die keine Abstraktion für die Ausnahmebehandlung besitzt, benutzt spezielle Werte als Ergebnis der Funktion, in der die Ausnahme ausgelöst wird. Andererseits soll aber die in Tl vorhandene Abstraktion `try ... end` zur Ausnahmebehandlung greifen.

Dazu dazu wird durch die C-Operation

```
void tmexcept_raiseWithGC(tsp_OID oidException, ...);
```

der Tycoon-Maschine eine Tl-Ausnahme ausgelöst. Sie übernimmt den OID[11] einer Tl-Ausnahme als Parameter `oidException`. Die obene genannten Ausnahmeabfangmechanismen werden dann in Tl wirksam und eine notwendige Stack-Bereinigung bis zur Abfangebene findet dabei automatisch statt.

2.2.3 Integrative Schranken

Dem Einsatz von Tycoon-Tl als systemverschmelzende Sprache, die einerseits mehrere wichtige Applikationen in Form von Bibliotheken mitbringt und zum anderen die Integration weiterer Applikationssysteme über verschiedene Schnittstellen (vgl. Abschnitte 2.2.1,2.2.2) zuläßt, sind jedoch auch Grenzen gesetzt. Diese ergeben sich immer dort, wo die Modellierungsmittel des einen Systems zu stark von denen des anderen Systems abweichen.

In Tl wird mit Daten, Code und Threads (vgl. Abschnitt 2.1.1) modelliert. Wobei durch Polymorphismus und Funktionen als Werten erster Klasse eine hohe Ausdrucksmächtigkeit erlangt wird. Dies begünstigt die Abbildung semantischer Objekte eines Fremdsystems in das Tycoon-System im Rahmen einer Systemintegration. Lediglich die Darstellung von Code durch Funktionen mit einer anzahlig festen Parameterliste und einem einfachen Ergebnisparameter stellt eine scheinbare Einschränkung gegenüber Darstellungen von Funktionen bzw. weniger streng typisierter Systeme mit variabler Parameterliste oder mehrfachen Ergebnisparametern dar.

Problematisch dagegen ist die Erstellung eines bidirektionalen Gateways zwischen Tycoon-Tl und einem Fremdsystem, weil viele Systemumgebungen keine Ausdrucksmittel für Typsysteme höherer Ordnung, Threads oder Funktionen als Werte erster Klasse zur Verfügung stellen. In diesem Fall muß man sich bei der Abbildung auf die Ausdrucks- bzw. Modellierungsmittel des weniger mächtigen Systems beschränken.

2.3 Ausgewählte Systemintegrationen

Im folgenden werden fünf ausgewählte Gateways und Schnittstellen des Tycoon-Systems zu externen Systemen vorgestellt. Dabei sollen die wichtigsten Eigenschaften, Anforderungen und Leistungen untersucht werden. Außerdem sind die Integrationsbeispiele so gewählt, daß verschiedene Basiskommunikationsschnittstellen, Bindungstechniken, Verfahren zur Behandlung flüchtiger C-Halden-Objekte und Eigenschaften, wie Persistenz, Mobilität und Kommunikationsrichtung, zur Geltung kommen.

[11] OID = Object IDentifier; alle Objekte des Tycoon-Systems werden eindeutig durch ihren OID identifiziert.

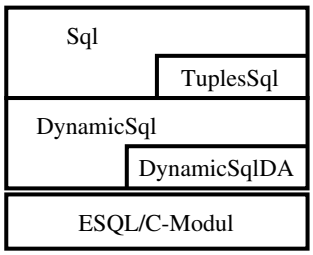


Abbildung 2.2: Modulhierarchie der Tycoon-SQL-Schnittstelle

2.3.1 SQL

Die SQL-Schnittstelle aus [Kor93][12] ermöglicht die Ausführung von SQL-Befehlen aus dem typsicheren Tycoon-Kontext. Sie erfüllt die Anforderung:

- Transfer von variablen Daten zur Datenbank
- Transfer mengenwertiger Daten
- Repräsentation von NULL-Werten
- Fehlererkennung

Die Leistungen der Schnittstelle bestehen aus den durch Typoperatoren dargestellten Objekten `Type`, `Table` und `Query` mit jeweils zugehörigen Funktionen zur Konstruktion dieser Objekte beziehungsweise zur Durchführung von Operationen auf den Objekten. Wenn zum Beispiel eine durch `Query` repräsentierte SQL-Anfrage mit der Operation

select(E <:**Ok** query :Query(E)) :Iter.T(E)

durchgeführt wird, erzeugt sie eine Iteration der Ergebnismenge. Zusätzlich gibt es noch administratorische Funktionen, um Datenbankverbindungen aufzubauen, abzubauen und Transaktionen fortzuschreiben oder zurückzusetzen.

Die Realisierung der Schnittstelle umfaßt im Fall der Datenbank Ingres, wie in Abbildung 2.2 dargestellt, drei Schichten mit fünf Modulen:

Sql Die für den Applikationsprogrammierer sichtbare Schnittstelle.

TuplesSql Bidirektionale Umwandlung der intern durch Descriptoren dargestellten Tupel in Tycoon-Tupel.

DynamicSqlDA Zugriff auf Descriptoren des Dynamic-SQL und Tycoon-Repräsentation der SQL-Datentypen durch numerische Kodierung, Länge und Format.

DynamicSql Eine untypisierte Schnittstelle zur Funktionalität von Dynamic-SQL.

ESQL/C C-Funktionen, die die Funktionalität von Dynamic-SQL verfügbar machen, wie Anlegen, Beschreiben und Ausführen von SQL-Anweisungen, Cursoroperationen und Funktionen zum Zugriff auf Descriptoren.

Die Schnittstelle erzeugt während einer Verbindung Descriptoren repräsentierende flüchtige Datenobjekte auf der Prozeßspeicherhalde. Diese werden aber bei der Beendigung der Verbindung automatisch wieder gelöscht. Ein Stabilisieren des Tycoon-Objektspeichers während einer Verbindung

[12] Sie wurde ursprünglich für die Sprache Quest [Car90] entwickelt, wurde aber inzwischen an Tycoon angepaßt.

führt deshalb zu hängenden Referenzen. Das gleiche Problem tritt im Zusammenhang mit einer Code-Migration auf. Deshalb ist diese Schnittstelle weder persistent noch mobil, kann jedoch mit den Mitteln der transaktionalen Verwaltung für flüchtige Objekte aus [Kor95] sowohl persistent als auch mobil gemacht werden.

Die SQL-Anbindung bietet statische Typsicherheit unter Ausnutzung von parametrischem Polymorphismus, reflektivem Zugriff auf den Tycoon-Objektspeicher und einer explizit bei jeder Erzeugung eines `TYPE(E)` anzugebenden Tupleprobe vom Typ `E`:

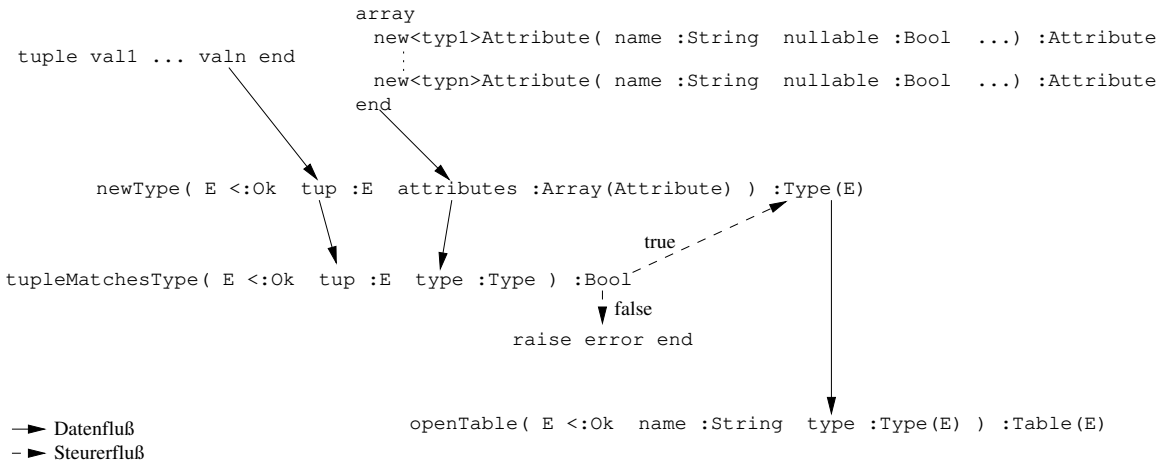
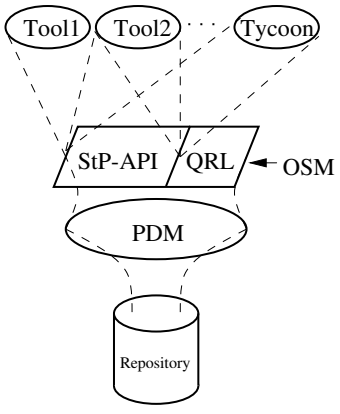


Abbildung 2.3: Funktionsverschachtelung beim Öffnen einer Tabelle

Die `TYPE(E)` konstruierende Funktion `newType()` ruft dazu `tupleMatchesType()`, die ihrerseits überprüft die Kompatibilität von `tup` und `type`, indem es die Tupeleinträge von `tup` aus dem Objektspeicher ausliest und abhängig vom entsprechenden Attribut in `type` Plausibilitätstests auf dem Objektspeicherformat durchführt. Dies ist notwendig, weil im übersetzten Tl-Programm keine Typinformation vorliegt und nur das Speicherformat überprüft werden kann. Das ist aber z.B. für verschiedene benutzerdefinierte Integertypen gleich. Daher beschränkt sich die SQL-Schnittstelle Tycoon-seitig auf die Unterscheidung zwischen `Int`, `Real` und `String`, während in SQL `Int1`, `Int2`, `Int3`, `Float4`, `Float8`, `Date`, `Money`, und `Varchar` zur Verfügung stehen.

2.3.2 Stp/OMT

In [Nie96] wird die Schnittstelle zur Case-Umgebung StP-OMT beschrieben. StP-OMT umfaßt neben einer Benutzeroberfläche, die verschiedene Werkzeuge zur Modellierung auf Basis der OMT[13] integriert, das *Object Management System*(OMS), das *Persistent Data Model*(PDM) und ein Repository zur persistenten Datenspeicherung. Das OMS implementiert das PDM im StP-Repository.
Der Zugriff auf die Objekte des PDM, wie z.B. *note*, *item*, *file* oder *link ref*, erfolgt wahlweise durch die Abfragesprache *Query and Reporting Language*(QRL) oder durch die StP-API-Programmierschnittstelle für C.
Die Tycoon-StP-Schnittstelle setzt auf die StP-API auf und besteht aus einer Bibliothek zum dynamischen Binden und einem Satz von Tycoon-Modulen. Die Module, wie *omsNote*, *omsFile* oder *omsLink*, implementieren jeweils eine Objektklasse des PDM.



Die von der Schnittstelle geforderten Eigenschaften, sind laut [Nie96]:

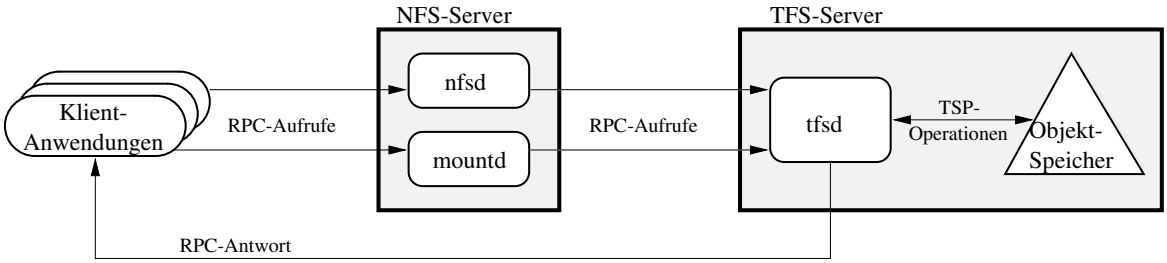


Abbildung 2.4: NFS-Anbindung aus [Gor95]

Persistenz Persistenz bedeutet die Objektspeicherung über die Lebensdauer des Erzeugerprozeß hinaus. Persistenz wird von der Schnittstelle nicht unterstützt. Zum einen gibt es keinen Mechanismus, der die automatische Erzeugung bzw. Zerstörung von flüchtigen Daten durchführt. Andererseits sind einige kompensierende Funktionen, die notwendig wären um flüchtige Heap-Objekte beim Operations-Logging freizugeben, nicht implementiert.

Typisierung Als Vorteile der Typisierung führt [Nie96] an, daß die in der C-Schicht teilweise untypisierten PDM-Objekte, in Tycoon durch Typoperatoren und parametrischen Polymorphismus typsicher sind. Außerdem können Sichtbarkeitsbereiche so gewählt werden, daß OMS-Funktionen nur im direkten Kontext ihrer Objekte erscheinen[14].

Flexibilität Flexibilität ist die Unabhängigkeit der StP-Anbindung von speziellen Anwendungen. Diese Eigenschaft zielt auf die Mächtigkeit der Anbindung an der Integrationsschnittstelle ab, die wichtig ist, wenn Tycoon als systemverschmelzende Sprache eingesetzt wird. Ein besserer Ausdruck hierfür wäre Generizität.

Wartbarkeit Unter Wartbarkeit wird ein geringer Aufwand bei dem Nachvollziehen von Änderungen der StP-API genannt. Allerdings ist die Wartungsaufwand für die C-Schicht als hoch angegeben.

In einer parallel zu der vorliegenden Diplomarbeit verlaufenden Arbeit(→[Nie97]), wird eine Modellierungs- und Simulationsbibliothek entwickelt. Diese soll eine Simulation und Entwurfsvalidierung der, mit den Möglichkeiten der *Harel State Charts*, dem dynamischem Modell in OMT, erzeugten Prozeßmodelle realisieren.

2.3.3 NFS

Die Arbeit [Gor95] beschreibt eine Anbindung des Tycoon-Objektspeichers an das SUN-NFS Dateisystem. Im Gegensatz zu der üblichen Vorgehensweise, die Funktionalität des Fremdsystems nach Tycoon abzubilden, um so Funktionen und Abläufe in Tycoon verfügbar zu machen, besteht die Leistung dieses Gateways darin, entfernten Zugriff von Standard-UNIX Werkzeugen auf den Tycoon Objektspeicher zu ermöglichen. Dazu sind die folgenden Komponenten und Dienste realisiert worden:

- Objektspeicherstrukturen in Tycoon, mit denen NFS-Dateisysteme in Objektspeichern realisiert werden.
- Umsetzung von NFS-Anfragen in TSP-Operationen auf dem Objektspeicher.
- Tycoon-Funktionen, die Tycoon-Datenobjekte auf Dateisystemobjekte des Objektspeichers abbilden.

[14] Ein Programmierstil der häufig methodenorientiert genannt wird, im Vergleich zur Objektorientierung, die allerdings weitere Spracheigenschaften, wie Vererbung und Kapselung voraussetzt.

Die Komponenten der Anbindung und ihr Zusammenspiel sind, wie in Abbildung 2.4 gezeigt, der TFS-Dämon, der Tycoon-Objektspeicher und die SUN-NFS Dämonen *mountd* und *nfsd*.

Der Zugriff aus der Tycoon-Umgebung erfolgt durch Operationen, die ein Dateisystem erzeugen bzw. löschen, Dateien in bestimmten Pfaden mit bestimmten Zugriffsrechten anlegen oder deren Inhalt als String zurückliefern.

2.3.4 INQUERY

INQUERY ist ein Information Retrieval System. Seine Aufgabe liegt in der rechneradäquaten Repräsentation von Textsammlungen und Auswertung von Anfragen zu deren Inhalten[15]. Dazu steht eine Ausführung von Anfragen über Stapelbetrieb, interaktiv oder durch eine Anwendungsprogrammierschnittstelle für die Programmiersprache C zur Verfügung.

In [Ste96] wird die Anbindung des INQUERY an Tycoon mittels statischen Bindens der Anwendungsprogrammierschnittstelle an die Tycoon-Maschine beschrieben. Aus der umfangreichen Bibliothek wurden Funktionen zum Zugang zu INQUERY-Datenbanken, zum Parsen von Anfragen, deren Auswertung und dem Zugang zu Dokumenten und Informationen der Textsammlung abgebildet.

Diese Funktionen nehmen zum Teil strukturierte Werte als Parameter entgegen, auf die über Selektorfunktionen zum Lesen und Beschreiben einzelner atomarer Einträge zugegriffen wird. In der notwendigen Schicht zwischen der INQUERY-API und dem Tyoon-System werden entsprechende strukturierte Werte angelegt, auf die die Selektorfunktionen zugreifen. Diese sind aber notwendig flüchtige Objekte im C-Heap und unterliegen nicht der Persistenz des Tycoon-Systems, obwohl in Tycoon zeitweilig Referenzen auf sie angelegt werden[16].

Als Vorteil der Abbildung strukturierter Werte über Selektorfunktionen, im Vergleich zu direkter Speichermanipulation durch das Tl-Modul `wordOp`, ergibt sich eine bessere Protabilität. Die Speicherdarstellung ist nicht mehr von bestimmten in Tycoon fixierten Annahmen abhängig.

Die INQUERY-Anbindung benötigt das flexible Typsystem, um eine bessere Differenzierung der Schnittstellendatentypen herzustellen. Außerdem wird durch die Subtypisierung `T <:String` generell zwischen geparsten Anfragen vom Typ `:T` und ungeparsten Anfragen vom Typ `:String` unterschieden.

Orthogonale Persistenz wird ausgenutzt zur langfristigen Speicherung von Anfrageergebnissen und zum Anlegen von Standardanfragen.

2.3.5 C++-Gateway

Das C++-Gateway aus [Gei95] nimmt eine Sonderrolle im Vergleich mit den zuvor vorgestellten Anbindungen ein. Alle bisherigen Anbindungen basieren auf der direkten Abbildung von Anwendungsinteraktionspunkten über ein Bindekonstrukt auf Werte und Typen des Tycoon-System. Das C++-Gateway hingegen kann eher als generischer Gateway-Generator verstanden werden. Motiviert durch die Anbindung der portablen C++-Klassenbibliothek Starview bestand die Hauptanforderung in der Integration graphischer Dienste in die persistente Umgebung Tycoon.

Der sehr allgemein gehaltene Ansatz geht von einer abstrakten Definition für ein Gateway aus:

> Ein *Gateway* ist eine Software-Komponente eines Systems, die die erforderlichen Bindungen gehörend zu *einer* externen Bibliothek [...] enthält. Es besteht im allgemeinen aus einer Menge modular zusammenhängender Teil-Gateways. Ein *generisches* Gateway ist durch *Schnittstellenbeschreibungen* parametrisierbar und generiert als Instanzen spezialisierte (Teil-)Gateways.

Dabei wird zwischen L_{intern}, der Systemsprache, und L_{extern}, der Sprache der externen Bibliothek, unterschieden (→[SM93]). Die Aufgabe des Gateways ist die transparente Bereitstellung entsprechender Bindungen von Objekten der Sprache L_{extern} in den programmiersprachlichen Raum von L_{intern},

[15] Im Vergleich zu herkömmlichen Datenbanken erlauben Information Retrieval Systeme unscharfe oder inhaltsbasierte Anfragen.

[16] [Ste96] weist darauf hin, betrachtet dieses Problem aber nicht als Gegenstand der Arbeit.

Gateway	Schnittstellentechnik[a]	Basisschnittstelle[b]	strukturierte Par.[c]	Persistenz[d]	Mobilität[e]	Richtung[f]	Typsicherheit[g]
SQL	E	C	Z	E	L	C	E
Stp/OMT	E	C	Z	K	N	C	Ja
NFS	E	S	Z			S	Ja
Inquery	E	C	Z	E	L	C	Ja
C++-Gateway	C	C	Z	E	L	B	Ja
C-Calls	E	I	I	V	M	B	E

[a] E=Eingebettet, C=Codegenerierung
[b] C=C-Calls, S=Sockets, I=Intern
[c] Z=Zugriffsfunktionen, D=Direkter Speicherzugriff, O=Objektspeicherzugriff, I=Intern
[d] P=Persistent, L=Operations Logging, K=Keine Persistenz
[e] M=Mobil, L=Operations Logging, N=Nicht Mobil
[f] B=Bidirektional, S=Server, C=Client
[g] J=Ja, E=Eingeschränkt

Tabelle 2.1: Eigenschaften verschiedener Integrationsschnittstellen des Tycoon-Systems

so daß ein Programmierer der Sprache L_{intern} auf Basis derer Regeln bzgl. Syntax und Semantik entwickeln und von der Existenz externer Sprachen und Dienste abstrahieren kann.

Handelt es sich bei L_{intern} um Tl, dann sollen die typischen Vorteile, wie statische Typisierung, Funktionen höherer Ordnung und Polymorphismus erhalten bleiben. Aus [SM93] werden weitere Anforderungen an das Gateway übernommen:

1. Das Typsystem von L_{intern} muß mindestens die Mächtigkeit des Typsystems von L_{extern}[17] haben.

2. Es muß ein Werkzeug geben, das die Abbildung von Signaturen aus L_{intern} auf Signaturen aus L_{extern} leistet.

3. Es muß einen Mechnismus geben, um zur Laufzeit Bindungen an externe Objekte, insbesondere Funktionen, zu etablieren.

Die Architektur des Gateways umfaßt die Abbildung programmiersprachlicher Konzepte aus C++ alias L_{extern}, wie Typen, Klassen, Vererbung, Overriding und Callbacks, auf entsprechende Konzepte in Tl alias L_{intern}, zum Beispiel Typen, Typoperatoren, Module. Dazu ist die Schnittstellenbeschreibungssprache GDL[18] definiert worden, die im Vergleich zu C++-Headerdateien von allen Implementationsdetails abstrahiert und einen rein deklarativen Charakter hat.

Ein sogenannter C++-Server kann nun in GDL spezifiziert werden. Anschließend wird diese Spezifikation mit einem speziellen GDL-Compiler bearbeitet, der Tl-Schnittstellen, Tl-Module und C++-Stubs erzeugt.

2.3.6 Zusammenfassung

An den vorgestellten Integrationen externer Systeme in die Programmierumgebung Tycoon läßt sich erkennen, daß es kein Standardrezept zur Systemintegration gibt. Unterschiedliche Schnittstellen und

[17] Siehe auch Abschnitt 2.2.3
[18] Gateway Definition Language

verschieden mächtige Modellierungsprimitiven der Systeme untergraben ihre Offenheit. Dennoch lassen sich die verwendeten Protokolle und einige generelle Mechanismen vergleichend zusammenfassen (siehe Tabelle 2.1).

Es können zwei Bindungstechniken unterschieden werden. Einerseits können dem Tycoon-System Primitiven hinzugefügt werden, um eine direkte dynamische Bindung an ein externes Objekt herzustellen, wie im Beispiel der C-Call Schnittstelle (siehe Abschnitt 2.2.2). Andererseits kann ein Tl-Stub-Code Generator geschrieben werden, der aus der Analyse einer Schnittstellenbeschreibungsform Stub-Code zur Einbindung externer Objekte in neue Programme generiert, wie im Fall des C++-Gateways (vgl. Abschnitt 2.3.5). Erstere Technik verlangt den Einsatz reflektiver Mechanismen, wie dynamischer Typen und automorpher Werte, weil erst zur Laufzeit die Schnittstellensignaturen festgelegt werden und danach aktuelle Parameter gegen diese Signaturen typüberprüft werden müssen. Durch diesen Laufzeittyptest entsteht zusätzlich ein Performanznachteil.

Auffällig in Tabelle 2.1 ist der Zusammenhang zwischen dem Verwenden der C-Calls-Schnittstelle als Basisschnittstelle und der Persistenz respektive Mobilität über ein Operations-Log. Die Ursache hierfür findet sich in dem Umstand, daß in einer derartigen Anbindung meistens flüchtige Daten erzeugt werden, die eben nur Pseudo-Persistent oder -Mobil sind (vgl. Abschnitte 2.1.2,2.1.3).

Die Kommunikationsrichtungen sind weitgehend anwendungsabhängig. Hierbei nimmt das NFS-Gateway eine Sonderrolle ein, weil es als einziges ausschließlich in der Server-Rolle[19] agiert. Bei allen übrigen Gateways befindet sich die Tl-Anwendung wenigstens in der Client-Rolle. Das C++-Gateway und die C-Call-Schnittstelle arbeiten sogar bidirektional.

Die Typsicherheit innerhalb des Tycoon-Systems wird von fast allen Schnittstellen gewährleistet. Ausnahmen bilden die C-Call-Schnittstelle, die nur atomare Basistypen unterstützt, und das SQL-Gateway, welches nur eine eingeschränkte Teilmenge der SQL-Basistypen abbildet. Beide Schnittstellen sind durch einen Laufzeitbindemechanismus realisiert, verzichten aber auf den Einsatz dynamischer Typen (siehe oben). Die C-Call-Schnittstelle umgeht das Problem, indem zum Definitionszeitpunkt der Bindung keine Übereinstimmung zwischen Funktionstyp und Steuerstring garantiert wird. Das SQL-Gateway hingegen prüft eine Tupel-Signatur gegen einige eingeschränkte Informationen aus dem Objektspeicherformat einer Tupelprobe, was aber keine vollständige Typsicherheit gewährleistet (vgl. Abschnitt 2.3.1).

Als generelle Forderung an die Systemintegration im Sinn der Definition aus Abschnitt 1 mit Tycoon-Tl als „System-verschmelzende“ Sprache lassen sich folgende Eigenschaften ableiten:

Applikationstransparenz Externe Applikationssysteme sollen derart integriert werden, daß ein Entwickler keine Beschränkung der Applikationssemantik durch die Integrationsschnittstelle erfährt und einen möglichst umfassenden Zugriff auf alle semantischen Objektklassen der Applikationssystems erhält.

Typsicherheit Externe Systeme und deren semantische Objekte werden bei der Integration nach Tl auf abstrakte Datentypen abgebildet. Diese Abbildung ist so zu wählen, daß semantisch unterschiedliche Objekte des Applikationssystems in Tl typverschieden sind. Im Zweifelsfall wird also jede Objektklasse auf einen separaten abstrakten Datentyp abgebildet. Wenn die Abbildung von Objekten feinerer Granularität durchgeführt wird, muß sichergestellt werden, daß entsprechend passende semantische Konstrukte zur Verfügung stehen.

Generizität(Flexibilität) Eine Systemintegration stellt eine instanziierbare generische Schnittstelle dar. Eine bestimmte Anwendung wird damit nicht impliziert.

Datenunabhängigkeit(Wartbarkeit) Die Integrationsschnittstelle bildet semantische Objekte ab. Dabei abstrahiert sie die konkrete Repräsentation im Applikationssystem, so daß die Lokalität der Objekte im Applikationssystem erhalten bleibt.

[19]Die andere Richtung wird durch Kombination der Tl-Dateischnittstelle und einem Standard-UNIX-Remote File System abgedeckt.

Kapitel 3

Integrationsmöglichkeiten des SAP R/3 Systems

SAP R/3 ist ein portables, vollintegriertes, verteiltes Standardsoftwareprodukt. Die Leistung des Systems besteht nicht in der technischen Perfektion, die bei kurzfristigen Innovationszyklen der Informatik nur schwer aufrecht zu erhalten wäre. Vielmehr sind durch die betriebswirtschaftlichen Module (→[Wen95b]) Abstraktionen auf einer anwendungsnahen Ebene geschaffen worden, die die SAP R/3-Systemtechnik lediglich als Basisdienst nutzen und für die die Attribute Wirtschaftlichkeit, Machbarkeit und Kontinuität vordringlich sind. Deshalb versucht SAP mit dem Produkt R/3 wichtigen Trends der EDV-Branche zu folgen, die sich in den informationstechnischen Konzepten, die in Abschnitt 1 aufgeführt sind, widerspiegeln.

Nach [BEG96] besteht die aktuelle Herausforderung an die Anbieter betriebswirtschaftlicher Client/-Server-Standardlösungen in der nahtlosen Einbindung von Fremdsoftware. Insbesondere bestehen zukünftige Konzepte für betriebswirtschaftliche Standardsoftware nicht mehr aus monolithisch konstruierten Architekturen, sondern aus Software-Komponenten. In diesem Zusammenhang spielen die heutigen Anbieter von Standardsoftware die Rolle von Systemintegratoren, die nur einen Teil der Gesamtfunktionalität als Komponenten zur Verfügung stellen. Weitere Komponenten werden von Fremdanbietern hinzugefügt.

Welche Möglichkeiten SAP R/3 zur Integration von Fremdkomponenten bietet und inwieweit offene Schnittstellen des Systems eine Einbindung in kapselnde Systeme fördern, was in dieser Arbeit in Kapitel 5 speziell am System Tycoon durchgeführt wird, ist neben einer groben Systembeschreibung der Gegenstand dieses Kapitels.

Dazu wird ein Überblick der Systemarchitektur gegeben, welcher der Einführung spezieller Begrifflichkeiten im SAP R/3 dient. Auf dieser Grundlage lassen sich die semantischen Objekte der Programmierung und Applikationsschicht leicht identifizieren. Aus einer Untersuchung der offenen Schnittstellen anhand der Systemarchitektur wird der *Remote Function Call*(RFC) als diejenige Schnittstelle mit den besten Möglichkeiten zum Zugriff auf die semantischen Objekte gefunden. Die Kommunikationsendpunkte des RFC sind Funktionsbausteine in SAP R/3. Die Analyse aller wichtigen Aspekte der Funktionsbausteine folgt, wie zum Beispiel der Programmiersprache ABAP/4, dem Datenkatalog und Typsystem, der Datenbanktransaktionen und letztendlich der Client/Server-Unterstützung durch Funktionsbausteine und den RFC.

Das System SAP R/3 hat jedoch einen Umfang, dessen vollständige Beschreibung den Rahmen dieser Arbeit sprengen würde, so daß auf [SAP96a] verwiesen wird, das eine umfassende Dokumentation des gesamten SAP R/3-Systems enthält. Die oben beschriebene Vorgehensweise für dieses Kapitel folgt dem Anspruch aus einem groben Überblick schnell zu den entscheidenden Tatsachen zu gelangen und diese dann im Detail vorzustellen.

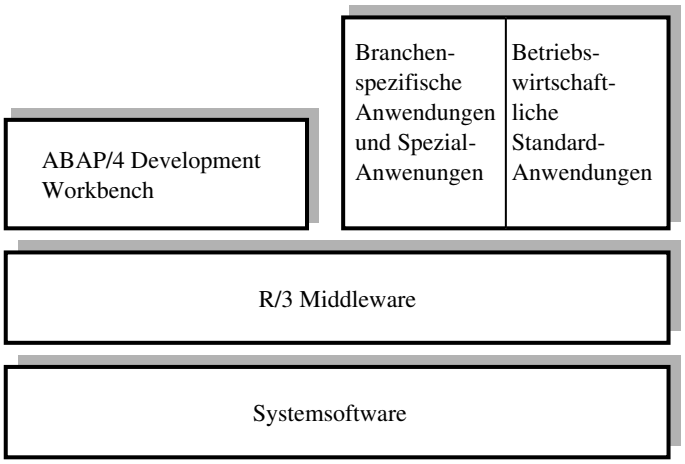


Abbildung 3.1: Grobarchitektur des SAP R/3 Systems aus [BEG96]

3.1 Modellarchitektur des Systems

SAP R/3 enthält als Standardsoftwaresystem für betriebwirtschaftliche Anwendungen Funktionalität aus den Bereichen der Betriebssysteme, Datenbanken, Benutzerschnittstellen und Applikationen. Wie in Abbildung 3.1 dargestellt, kann zur groben Erfassung zwischen den systemtechnischen Basisdiensten und einer darauf aufbauenden Anwendungsschicht unterschieden werden. Erstere umfassen die Systemdienste und die R/3-Middleware, die in heterogenen EDV-Umgebungen eine homogene Schnittstelle für alle aufsetzenden Dienste und Programme darstellt. Die Anwendungsschicht setzt sich ihrerseits aus der ABAP/4-Workbench und den eigentlichen betriebswirtschaftlichen Standard- und Benutzeranwendungen zusammen. Im folgenden werden zunächst die Dienste der R/3-Middleware vorgestellt.

3.1.1 Dienste und Prozeßarchitektur der R/3-Middleware

Die Benutzerschnittstelle ist das SAPGUI[1]; eine graphische Terminalanwendung, die unter Ausnutzung von Diensten der jeweilig vorliegenden graphischen Benutzeroberfläche Daten ausgibt und Tastatur- bzw. Mauseingaben entgegennimmt. Es werden verschiedene graphische Benutzeroberflächen[2] unterstützt (→[Wen96]). Dabei bietet das SAPGUI eine Untermenge der üblichen semantischen Kontrollelemente an, die dafür aber für alle Präsentationsplattformen einheitlich unterstützt werden. Im Bezug auf die Kommunikation im Netzwerk hat das SAPGUI einen Vorteil gegenüber dem reinen X-Protokoll[3], da nur die semantischen Informationen der Darstellungselemente übertragen werden, nicht jedoch die pixelorientierte Darstellung insbesonders datenintensive Bitmaps selbst. So werden pro Dialogschirm nur etwa ein bis zwei Kilobyte Daten übertragen (→[BEG95] Seite 95), eine Eigenschaft, die das SAPGUI auch für die Benutzeranbindung in WANs[4] tauglich macht.

Applikationsdienste beinhalten die eigentliche Verarbeitung. Sie sind konkurrent ablaufende Betriebssystemprozesse, sogenannte *Workprozesse* mit unterschiedlichen Aufgaben, die zur *Instanz*, dem Applikationsserver, des SAP-Systems zusammengefaßt werden. Es gibt Workprozeßtypen für Dialogverarbeitung, Verbuchung, Batch-Verarbeitung, Drucker-Spooling und für die Sperrverwaltung. Dabei muß pro Instanz wenigstens ein Dialog-Workprozess existieren, damit der Zugriff auf das System

[1] SAP Graphical User Interface

[2] OSF/Motif, Presentation Manager, Windows 3.1, Apple Macintosh

[3] Würde das SAPGUI mit den Darstellungselementen eines X-Toolkits arbeiten, hätte es diesen Vorteil nicht. Dann würde aber die Unabhängigkeit von der graphischen Benutzeroberfläche wegfallen.

[4] Wide Area Networks

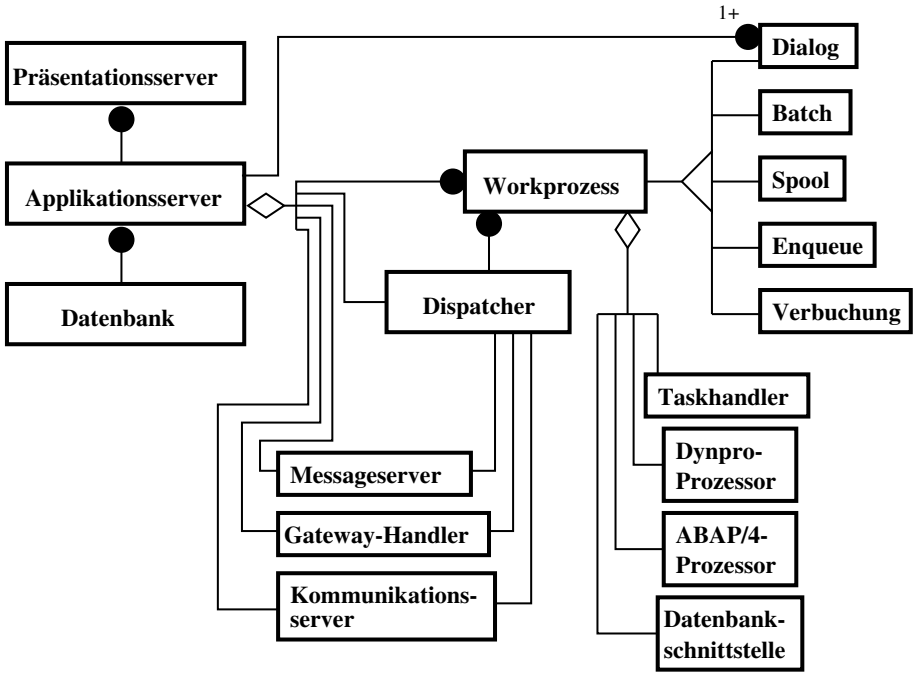


Abbildung 3.2: Dienstehierarchie des SAP R/3-Systems

gewährleistet ist. Die Zahl und Art der Workprozesse wird in einem Systemprofil festgelegt und bleibt für die Dauer des Betriebs konstant.

Workprozesse sind nicht einem bestimmten Benutzer zugeordnet. Sie sind in ihrer Art und ihrer Anzahl Teil der SAP R/3-Instanz und erfüllen die durch ihren Typ bestimmte Aufgabe. Die Aufgabenverteilung an die Workprozesse nimmt der in jedem Applikationsserver genau einmal vorhande *Dispatcher* vor. Er verwaltet eine Warteschlange aus anstehenden Aufgabe und teilt diese frei werdenden Workprozessen zu. Dadurch kann z.B ein Dialog-Workprozess mehrere Benutzer bedienen.

Die Koordination innerhalb eines Workprozesses wird durch den *Taskhandler* erledigt. Er ruft zum Beispiel den ABAP/4-Prozessor oder den Dynpro-Prozessor auf, wenn ABAP/4-Code abzuarbeiten bzw. Dialoge durchzuführen sind.

In den Dispatcher ist eine APPC-Schnittstelle integriert. Sie wickelt in heterogenen Kommunikationsumgebungen die Kommunikation über das *Kommunikations-Gateway* [5] ab (→[CCo96]). Es arbeitet als eigenständige Applikation in Kooperation mit den restlichen Komponenten der Instanz. Betroffen sind davon insbesondere Übergänge zwischen TCP/IP-basierten Netzwerkumgebungen für R/3 und SNA-Netzwerke mit R/2 Komponenten.

Eine weitere externe Applikationen einer SAP R/3-Instanz ist der *Kommunikationsserver*. Der Kommunikationsserver führt den dokumentenbasierten Datenaustausch, wie elektronische Post, EDI[6] und Telematik Dokumente, wie Telefax, Teletex und Telex, durch (→[TSC96]). Allerdings gehört zu jedem dieser Dienste auch eine SAP interne Komponente, die die richtige Bearbeitung und den Austausch mit dem Applikationsserver durchführt.

Der *Messageserver* koordiniert in verteilten SAP R/3 Umgebungen entfernt auszuführende Aufgaben. Dazu gehören Verbuchung, Batch-Jobs, Sperrverwaltung und der Austausch von Aufträgen. Zusätzlich überprüft er die angeschlossenen Applikationsserver, ermittelt denjenigen mit geringster Prozessorbelastung und ist damit der Basisdienst der Lastverteilung einer verteilten SAP R/3-Instanz.

[5] In der Literatur wird es unterschiedlich benannt: Gateway-Handler, CPIC-Handler, CPIC-Server, ...

[6] Elektronic business Data Interchange – Ein Standard, der Geschäftsdokumente normt, so daß eine Netzwerkübertragung und automatische Bearbeitung ermöglicht wird.

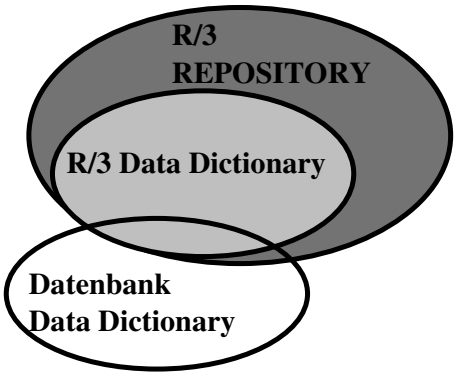


Abbildung 3.3: SAP R/3 Repository (Darstellung aus [WHSH95])

Es gibt in verteilten Umgebungen des SAP R/3-Systems immer genau einen Messageserver.

Die Datenbank speichert persistente Daten des Systems, wozu SAP-Tabellen[7], ABAP/4-Programmen, SAP-Transaktionen, Stammdaten und Metadaten gehören. Letztere hat SAP in dem *Data Dictionary*, dem Datenkatalog des Systems, zusammengefaßt. Insbesondere enthält die Datenbank aber auch noch einen eigenen Datenkatalog, so daß sich der in Abbildung 3.3 gezeigte Zusammenhang ergibt.

Wegen des relationalen Datenmodells können nur RDBMSe[8] verwendet werden. Der Zugriff findet entweder durch Open-SQL oder Native-SQL statt. Die Open-SQL-Schnittstelle realisiert einen von konkreten SQL-Implementationen abstrahierenden Zugriff auf die Datenbank. Dadurch können verschiedene Datenbanken mit unterschiedlichen SQL-Implementationen verwendet werden. Die Verwendung datenbankspezifischer SQL-Dialekte ist zusätzlich durch Native-SQL, das den jeweiligen datenbankspezifischen SQL-Dialekt unterstützt, realisiert. Allerdings erkauft sich der Entwickler bei der Verwendung von Native-SQL einen Nachteil bezüglich der Portabilität seiner Anwendung, da er damit von dem gewählten SQL-Dialekt abhängig ist.

Zusammenfassend ergibt sich die in Abbildung 3.2 dargestellte Architektur der Systemdienste in den Schichten Präsentation mit dem SAPGUI, Applikation mit ihren Workprozessen und dem Dispatcher und Datenbank als Diensterbringer für die Applikationsschicht. Diese Architektur entspricht der Architektur moderner Informationssysteme aus Kapitel 1 mit ihren drei Schichten Interaktion, Applikation und Datenbank.

Im folgenden wird das semantische Modell mit seinen Primitiven als Grundlage für die anschließende Untersuchung der offenen Schnittstellen eingeführt.

3.1.2 Semantische Objekte

Die Voraussetzung zur Integration des SAP R/3 in die Programmierwerkbank Tycoon ist die Erfassung seiner semantischen Objekte. Sie sind diejenigen Bausteine, die innerhalb von Tycoon repräsentiert werden müssen, um einen möglichst vollständigen und damit transparenten Zugriff auf das System SAP R/3 aus Tycoon zu erlangen.

Im folgenden werden diese semantisch abgrenzbaren Hauptkonstrukte des SAP R/3, SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen, vorgestellt:

SAP-Tabellen sind die persistenten Datenobjekte des SAP R/3. Sie stellen einen variablen Mengenwert dar und haben weitgehend die gleichen Eigenschaften, wie die Datenbanktabellen üblicher RDBMs. SAP-Tabellen sind in Abschnitt 3.2.2 beschrieben.

[7] Im Unterschied zu gewöhnlichen Datenbanktabellen
[8] Relational Database Management System

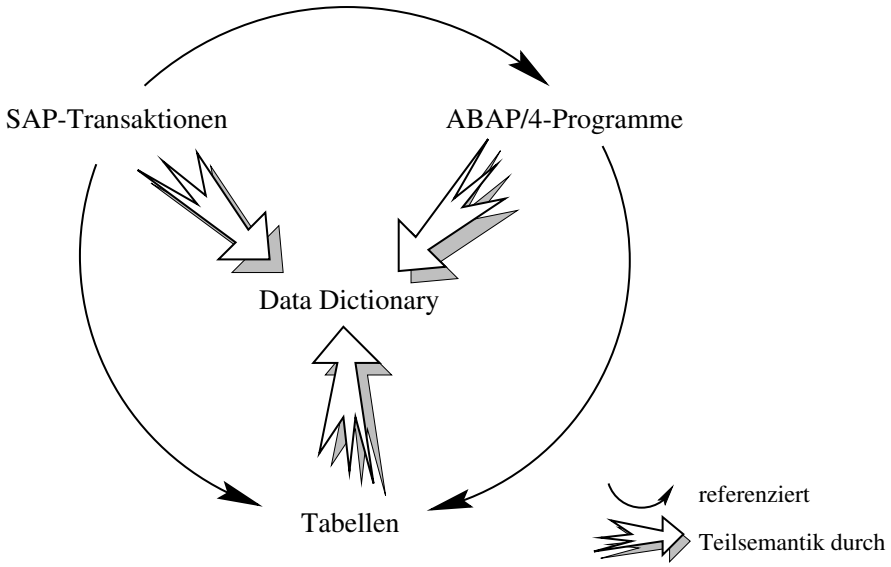


Abbildung 3.4: Semantische Objekte des SAP R/3

ABAP/4-Programme können in SAP verschiedenartige Objekte, wie z.B. *Report*, *Funktionsgruppe* oder *Modulpool*, sein. Ihnen gemeinsam ist, daß sie

- einen Namen haben.
- in der Programmiersprache ABAP/4 geschrieben sind (vgl. Abschnitt 3.2.3).
- durch Includes, Unterprogramme und Makros strukturiert sind.
- einen dynamischen Kontext haben , der aus Laufzeitvariablen, einem Programmzähler, SAP- und Datenbank-Tabellen und Shared Memory-Bereichen besteht.

Damit ähneln ABAP/4-Programme in herkömmlichen 3GL-Sprachen geschriebenen Programmen. Allerdings sind sie nur in SAP R/3 ablauffähig und durch Abhängigkeiten zum Data Dictionary und weiteren Repository-Objekten fest in die SAP R/3-Umgebung eingebunden.

SAP-Transaktionen sind logisch abgeschlossene Kommunikationsstränge mit dem System, wie zum Beispiel betriebswirtschaftliche Handlungen. Derartige Kommunikationsstränge bestehen aus mehreren aufeinanderfolgenden Bildschirmen mit ihren Interaktionsmöglichkeiten. Dieses Konzept ist technisch durch *Dynpro*s realisiert, die bestehend aus einem *Screen*, einer *Ablauflogik* und einem *Modulpool*[9], jeweils einen Interaktionsschritt kapseln:

Screen Ein Screen ist die Bildschirmdefinition für einen Dialogschritt. Das ist ein rechteckiger Bereich, der Eingabe- und Bedienelemente enthält. Dazu gehören laut [BEG96]: Rollbare Ein-/Ausgabefelder, *Push Button*s, *Radio Button*s, *Check Box*en, Rahmen und *Table Control*s.

Ablauflogik Die Ablauflogik ist eine textuelle Beschreibung des Kontrollflußes für die Dynpro-Verarbeitung. Sie enthält im wesentlichen die beiden Bereiche *Process Before Output*(PBO) und *Process After Input*(PAI). In ihnen wird in einer ABAP/4-ähnlichen Syntax beschrieben, welche Daten in die Felder des Screens zu schreiben sind, bevor dieser angezeigt wird, bzw. auf welche Weise Eingaben eines Benutzers in Eingabefelder des Screens zu verarbeiten sind.

[9] Tatsächlich können noch weitere Objekte zum statischen Kontext eines Dynpros gehören, wie Menü, Symbolleiste, Funktionstasten und Drucktastenbelegung. Diese sind allerdings nicht zwingend notwendig.

Modulpool Ein Modulpool ist ein ABAP/4-Programm, daß die Module enthält, die aus der Ablauflogik aufgerufen werden. Module sind, semantisch gesehen, Unterprogramme der Ablauflogik.

Durch die Verkettung mehrerer Dynpros entstehen komplette Dynpro-Ketten oder Kommunikationsstränge[10] mit dem System. Die Einstiegsdynpros dieser Kommunikationsstränge können im SAP-System vierstelligen Transaktionscodes zugeordnet werden. In der Regel bezeichnet der Begriff *SAP-Transaktion* nur einen Transaktionscode (→[WAT96]).

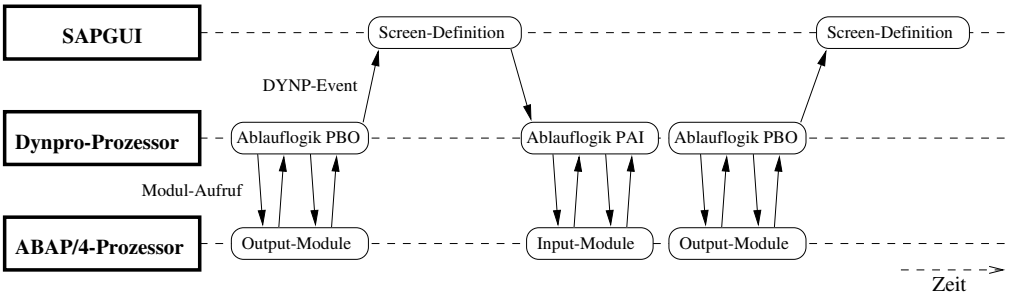


Abbildung 3.5: Ereignispfade der SAP-Transaktionen-Verarbeitung

Der dynamische Ablauf derartiger SAP-Transaktionen ist in Abbildung 3.4 aufgezeichnet. Ausgehend von der Ablauflogik des Einstiegs-Dynpros wird der PBO-Abschnitt durch den Dynpro-Prozessor des ausführenden Dialog-Workprozesses ausgeführt, um den Screen vorzuinitialisieren. Dabei werden verschiedene ABAP/4-Module aufgerufen, die ihrerseits durch den ABAP/4-Prozessor des Workprozesses ausgeführt werden. Danach generiert der Dynpro-Prozessor einen DYNP-Event, ein Datenobjekt, das eine vollständige Beschreibung eines Screens enthält, der dem SAPGUI zugesandt wird. Nun kann der Benutzer durch die Bearbeitungsmöglichkeiten des SAPGUIs Veränderungen an dem Screen-Inhalt vornehmen. Z.B. kann er Daten in Eingabefelder eintragen, Checkboxen aktivieren oder Radiobuttons auswählen. Wenn er dann durch eine Aktion die Beendigung der Bildschirminteraktion auslöst, wie z.B. beim Betätigen eines Druckknopfes, wird der Screen-Inhalt wieder in einen DYNP-Event geschrieben und dem ausführenden Dialog-Workprozess zurückgesandt. Nun beginnt der Dynpro-Prozessor mit der Abarbeitung des PBO-Abschnitts der Ablauflogik des Dynpros und ruft dabei entsprechende ABAP/4-Module auf. Danach wird ein neuer Auftrag für die Bearbeitung des Nachfolgedynpros in der Warteschlange des Dispatchers generiert.

Zum Zeitpunkt der Erstellung eines Dynpros wird ein Nachfolgedynpro festgelegt. Durch spezielle ABAP/4-Anweisungen kann ein Kommunikationsstrang jedoch beendet werden, auf andere Dynpros des akuellen Modulpools umgeleitet werden oder es kann eine neue SAP-Transaktion, zu der ein neuer Modulpool gehört, aufgerufen oder als Untertransaktion ähnlich einem Unterprogramm eingeschoben werden. Daher entscheidet sich der tatsächliche Ablauf einer SAP-Transaktion erst zur Laufzeit.

SAP-Transaktionen sind die wichtigsten semantischen Objekte in SAP R/3. Abgesehen davon, daß die interaktive Bedienung des SAPGUIs schon beim Einloggen in das System angefangen ausschließlich über SAP-Transaktionen abläuft befindet sich ein erheblicher Teil der Logik der betriebswirtschaftlichen Anwendungsmodule[11] in den SAP-Transaktionen. Die Modellierung von SAP-Transaktionen wird durch den Entwurf sogenannter *Ereignisgesteuerter Prozeßketten* (EPK) vorgenommen.

[10] Im Kontext anderer Arbeiten, wie z.B. den *Tycoon Business Conversations* (→[Joh95]), werden Kommunikationsstränge als Konversationen bezeichnet

[11] Die Anwendungsmodule sind wohl unterschieden von ABAP/4-Modulen eines Modulpools und dürfen nicht verwechselt werden.

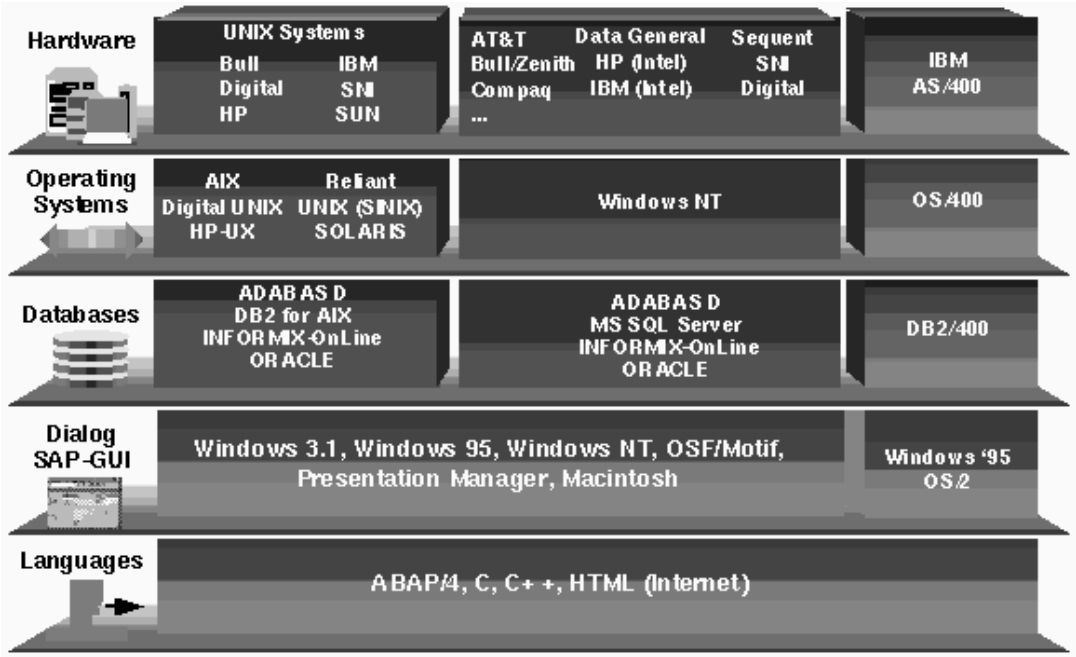


Abbildung 3.6: Offentheit von SAP R/3 (Abbildung aus [SAP96c])

3.1.3 Offene Schnittstellen

Die Offenheit des Softwaresystems SAP R/3 unterliegt den beiden Sichten *Portabilität* und *Integrativität* (vgl. Abschnitt 1). Erstere umfaßt alle Schnittstellen der SAP R/3-Middleware, die eine Integration verschiedener Fremdsystemen oder die Verwendung unterschiedlicher Betriebssystemplattformen ermöglicht. Einen Überblick über angestrebte Offenheit beziehungsweise integrierte Fremdsysteme und Dienste gibt die Abbildung 3.6 aus [SAP96c].

Besonders wichtig im Zusammenhang mit der evolutionären Gestaltung (vgl. Abschnitt 1) von Informationsystemen sind die integrativen Schnittstellen, die SAP R/3 selbst zum Diensterbringer machen. Diese Möglichkeit, die wesentlich für eine Integration in die Programmierumgebung Tycoon ist, siehe Kapitel 5, wird unter Berücksichtigung der vorgestellten Modellarchitektur des SAP R/3 im folgenden durch offene integrative Schnittstellen auf den drei Ebenen der Datenbank, des Applikationsservers und der Präsentation diskutiert.

Entfernter Datenbankzugriff

Der entfernte Zugriff auf das Datenbanksystem mit dessen Datenbanktabellen, ist grundsätzlich möglich. Die Applikationsdienste der SAP-Software selbst nutzen die Datenbank über RDA-Mechanismen[12], indem sie durch die Datenbankschnittstelle, *Remote-SQL*-Kommandos an die dem System angeschlossene Datenbank senden (→[BEG96]).

Für Fremdsysteme ist ein solcher Datenbankzugriff nicht empfehlenswert. Zwar können die Tabellen des SAP-Systems eingesehen, verändert oder gelöscht werden, jedoch ist die Gefahr Integritätsregeln der SAP-Applikations- und Präsentationsschicht zu verletzen sehr hoch, da diese im R/3-Data Dictionary durch ein mehrschichtiges Typsystem, Fremdschlüsselbeziehungen oder andere Abhängigkeiten festgehalten sind. Zudem könnten auch R/3-Systemdaten verändert werden, was zu unvorhersehbaren Effekten, einer Funktionsuntüchtigkeit des SAP-R/3 oder im schlimmsten Fall zu nicht erkannten betriebswirtschaftlichen Falschberechnungen führen kann.

[12] Remote Database Access

Die Applikationsschicht

Die SAP R/3-Applikationsschicht enthält die in Abschnitt 3.1.2 beschriebenen semantischen Objekte SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen. Sie werden durch die Prozeßarchitektur dieser Schicht erzeugt, interpretiert und verwaltet. Ihre offenen Schnittstellen umfassen die Dateischnittstelle, einen Mechanismus zur automatischen Datenübernahme, die Punkt-zu-Punkt Kommunikationsschnittstelle CPI-C, den darauf aufbauenden Remote Function Call, die Business APIs, elektronische Post und eine EDI-Schnittstelle. Sie werden im folgenden beschrieben, wobei jeweils Bezug auf die Fähigkeit der Schnittstelle genommen wird, die oben genannten semantischen Objekte SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen zu manipulieren:

Dateischnittstelle ABAP/4-Programme können über eine eingebaute Programmierschnittstelle Betriebssystemdateien lesen und schreiben. Dazu stehen folgende Anweisungen (→[SAP96a]) zur Verfügung (→[SAP96a]):

`OPEN DATASET` Öffnen einer Datei des unterliegenden Betriebssystems.

`TRANSFER TO` Schreiben eines Datensatzes.

`READ DATASET` Lesen eines Datensatzes vorgegebener Länge.

`CLOSE` Schliessen einer Datei.

Um einen Zugriff auf die semantischen Objekte des SAP R/3 über Dateien zu realisieren, muß ein Konzept zur Datenkodierung und Kontrollflußsteuerung als ABAP/4-Programm erzeugt werden, das dann über die Hintergrundverarbeitung gesteuert ständig läuft. In regelmäßigen zeitlichen Abständen würde dieses ABAP/4-Programm eine ausgezeichnete Datei überprüfen, die Synchronisations- und Befehlsinformationen in Form eines Ringpuffers enthält. Auf diese Weise kann dann ein SAP-externes Programm Daten und Befehle asynchron in die ausgezeichnete Datei stellen, aus der das ABAP/4-Programm sie zeitlich versetzt wieder herauslesen würde.

Die Verwaltung einer solchen Ringpufferdatei stellt zusätzlichen Aufwand dar. Die Verarbeitung würde ständig durch die Latenzzeiten der Hintergrundverarbeitung verzögert. Außerdem sind damit noch nicht die Probleme der Authentisierung und der Verteilung gelöst. Somit bietet die Kommunikation über Dateien nur sehr schlechte Unterstützung zur Systemintegration und ist daher ungeeignet.

Direkte Programminteraktion Durch CPI-C und den RFC stellt SAP Werkzeuge für die ortstransparente Interaktion zwischen ABAP/4-Programmen zur Verfügung. Die Grundlage dieser Kommunikation ist die Advanced Program to Program Communication(APPC)-Schnittstelle aus der System Network Architecture(SNA) von IBM.

SNA wurde 1974 von IBM als Netzwerkstandard entwickelt. Es wurde eine siebenschichtige Architektur, ähnlich dem zwischen 1977 und 1984 durch ISO/OSI spezifizierten Referenzmodell (→[Ste92]), festgelegt, in die sich entsprechende Protokolldienste einpassen. Wie in Abbildung 3.7, lassen sich die sieben SNA-Schichten grob in vier funktionelle Einheiten unterteilen: Endanwender, Logische Einheit, Wegsteuerung und Datenübertragung.

Ein Endanwender ist entweder ein realer Terminal-Anwender oder ein Anwenderprozeß[13]. Er benutzt zur Kommunikation eine logische Einheit(LU – Logical Unit). Die logische Einheit ihrerseits nutzt die Dienste einer physikalischen Einheit(PU – Physical Unit). Es gibt verschiedene Versionen der LUs und PUs zur Unterstützung unterschiedlicher Kommunikationsvarianten. Im Kontext von SAP ist die LU6.2 wichtig, die eine PU2.0 oder PU2.1 voraussetzt. Sie erlaubt die Kommunikation zwischen Anwenderprozessen. Das Dienstleistungsangebot der LU6.2 wird als APPC-Schnittstelle bezeichnet.

APPC spezifiziert sogenannte Konversationsverben, die die Semantik der APPC-Dienste beschreiben (→[Woh88]). Allerdings wird nicht vorgeschrieben, wie ein konkreter Aufruf eines

[13] In einer von IBM geprägten Terminologie heißen Anwenderprozesse Transaktionsprogramme (→[IBM88]).

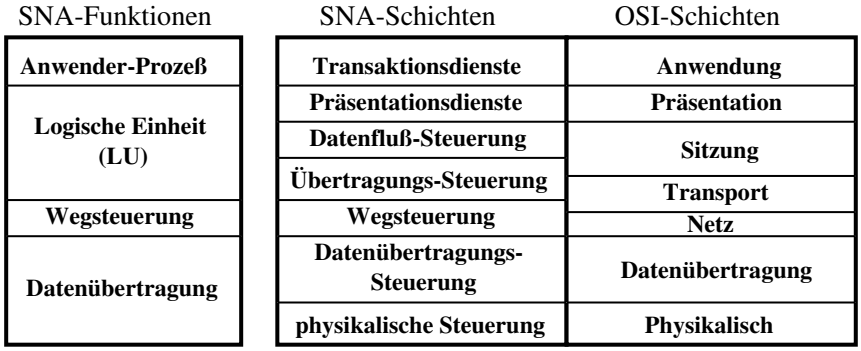


Abbildung 3.7: Die sieben Schichten des SNA-Modells und deren Zuordnung zum OSI-Modell (→[Ste92])

Dienstes syntaktisch aussieht. Dieser Umstand wird durch die Common Programming Interface – Communication(CPI-C)-Schnittstellendefinition behoben (→[IBM88]). CPI-C legt einen konkreten Befehlssatz fest, der die Konversationsverben des APPC implementiert.

Eine Kommunikation über APPC wird als *Konversation* bezeichnet. Solche Konversationen werden im Halbduplex-Verfahren durchgeführt, bei dem zu jedem Zeitpunkt genau ein Kommunikationsteilnehmer sendeberechtigt ist, während der andere empfangsberechtigt wartet. Eine Beispielkonversation ist in Abbildung 3.8 wiedergegeben:

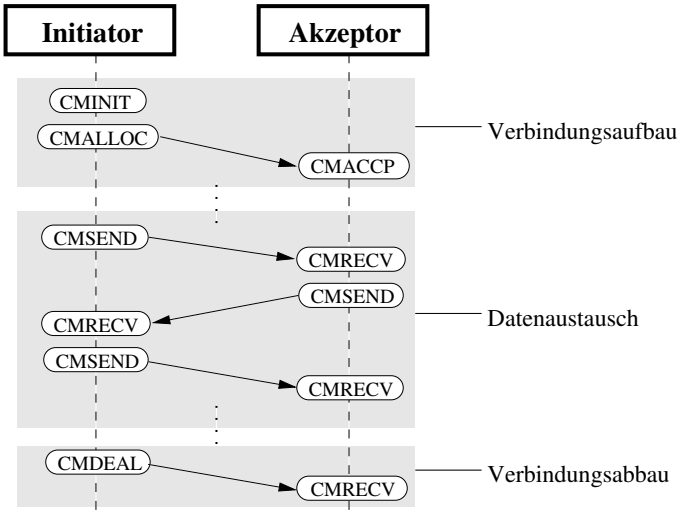


Abbildung 3.8: Unidirektionale CPI-C-Komunikation

APPC verläßt sich bei der Kommunikation vollständig auf die unterliegende Wegsteuerung. Daher kann diese Schnittstelle vom Prinzip auch auf anderen Netzwerken als SNA-konformen genutzt werden, soweit die der Wegsteuerung und unterliegenden Dienste entsprechenden Basismechanismen durch das Netzwerk zur Verfügung gestellt werden (→[Woh88]). Diese Eigenschaft nutzt SAP aus, indem CPI-C als einzige Basiskommunikationsschnittstelle für Netzwerke auf Prozeßebene, die entspricht ABAP/4-Programmen, genutzt wird. Alle ABAP/4-Programme sind deshalb ohne Änderung in den Umgebungen SNA und TCP/IP nutzbar. Außerdem bietet SAP Programmbibliotheken zur CPI-C-Kommunikation für verschiedene externe Systeme an, so daß CPI-C eine offene Schnittstelle zum SAP-System bildet.

Der **RFC** ist eine Leistungserweiterung der CPI-C-Schnittstelle im SAP R/3-System. Er gestattet den Aufruf lokaler oder entfernter Funktionen[14]. Im Gegensatz zur bidirektionalen CPI-C-Kommunikation orientiert sich der RFC am Client/Server-Modell mit seinen Service-Funktionen (Funktionsbausteinen), die durch ein Client-Programm aufgerufen werden (→[BEG96, CDK94]). Während das Protokoll für die Steuerung der Kontrollflüsse mit CPI-C explizit ausprogrammiert werden muß, wird es durch die Aufrufsemantik des RFC impliziert. Der Dienstnehmer, also der Aufrufer einer Service-Funktion, übergibt einen Aktualparametersatz im Funktionsaufruf. Die Konvertierung und Serialisierung der Parameter, die Anmeldung am System, sowie die Abwicklung der Kommunikation übernehmen Basisroutinen des RFC.

Wie für CPI-C gibt es auch für den RFC Programmierschnittstellen zum Anbinden externer Systeme. Eine RFC-Basisbibliothek dient der Anbindung von C-Programmen für verschiedene Hardware-Plattformen. Ergänzend gibt es für MS-Windows basierte Umgebungen eine OLE-Einbettung (→[OLE96a]), die den RFC als Kommunikationsschicht benutzt. Ein OLE Automation Controller gestattet es externe Applikationen, die als *Automation Client* fungieren, mittels ABAP/4-Anweisungen, wie `CREATE OBJECT`, `FREE OBJECT` und `CALL METHOD`, aus SAP heraus anzusteuern (→[AOA96]). Umgekehrt bietet SAP einen OLE Automation Server an, der zum einen den Aufruf von SAP Service-Funktionen gestattet, zum anderen aber auch einige OLE Custom Controls(OCX) für SAP-Transaktionen und SAP-Tabellen enthält. Eine weitere Ergänzung ist der Visual RFC Service Provider (→[VRF96]). Er beinhaltet eine Sammlung von Standarddialogen, um RFC-Verbindungen aufzubauen, abzubauen und Kommunikationssitzungen durchzuführen und zu überwachen.

In Abschnitt 3.2.3 wird gezeigt, daß durch ABAP/4-Anweisungen zum Zugriff auf Tabellen, zum Aufruf von ABAP/4-Programmen und SAP-Transaktionen ABAP/4-Service-Funktionen vollständige Steuerungsmöglichkeiten für die drei semantischen Objektklassen des SAP R/3 bieten. Dadurch bieten sich Service-Funktionen[15] in SAP R/3 als Integrationsschnittstelle für eine Einbindung in Tycoon an. Die Ergänzungen durch OLE und den Visual RFC Service Provider können nicht genutzt werden. Erstens setzen sie eine MS-Windows kompatible Programmumgebung voraus, mit der Konsequenz einer sehr starken Einschränkung der Portabilitätseigenschaft. Und zweitens besitzt Tycoon keine offene Schnittstelle zu OLE/COM und kann daher nicht von Standardbenutzerdialogen profitieren (siehe Abschnitt 2.3).

Business Application Programming Interface(BAPI) Als einzige Integrationsschnittstelle zum Zugriff auf das SAP R/3-System bietet SAP seit Version 3.0 die BAPIs an (→[BAP97]). Sie stellen ein Bindeglied zwischen dem monolithisch vollintegriertem SAP R/3-System und den Anforderungen moderner objektorintierter verteilter Systeme, wie Java, Active-X, COM/DCOM oder CORBA (→[OHE96a, OLE96b]), dar.

In dem *Business Object Repository*(BOR) werden von SAP sogenannte *Business Objects*(BO) abgelegt, die als typische Vertreter der Objektorientierung in ihren Methoden und Attributen Funktionalität zum Zugriff auf bestimmte betriebswirtschaftliche Stammdaten bzw. betriebswirtschaftliche Datensätze kapseln. Dabei wird von SAP ein *top down*-Ansatz verfolgt, in dem die von SAP verstandenen Zusammenhänge betriebswirtschaftlicher Weltmodellierung[16] in BOs strukturiert werden. Die Funktionalitäten der BOs, repräsentiert durch ihre Methoden, werden als BAPIs bezeichnet. Sie sind sprachunabhängig. Diese BAPIs stellen die Zurgiffsschnittstelle auf die BOs eines SAP R/3-Applikationsservers dar. Durch diese zusätzliche Abstraktion von konkreten SAP-Tabellen, ABAP/4-Programmen und SAP-Transaktionen aud BOs ist die Programmierung neuer externer Applikationen nicht mehr von der Änderung SAP-interner Objekte und Zusammenhänge abhängig. So daß SAP mit den BAPIs einen Schritt weg von der vollintegrierten monolithischen R/3-Lösung hin zu einem komponentenorientierten System kommt, dessen Teile als vergröberte Objekte verstanden werden können.

[14] Funktionsbausteine im SAP-Applikationsserver.
[15] Funktionsbausteine
[16] Laut [BAP97] hat SAP die wahre Geschäftswelt verstanden.

Die BAPIs bieten für die Realisierung einer konkreten betriebswirtschaftlichen Modulintegration eine ausgezeichnete Technik. Die generische Integration des SAP R/3 in Tycoon hapert jedoch an drei Punkten:

- Die Integration wurde mit Version 2.2j begonnen, die noch kein BOR besaß.
- Es müßten zur Integration des betriebswirtschaftlichen Modells des BOR alle BAPIs in generischer Weise in Tycoon-Tl integegriert werden, was wegen ihrer Zahl von mehr als 170 BOs ein aufwendiges Unterfangen wäre.
- Tycoon-Tl besitzt zum Zeitpunkt der erstellung dieser Arbeit keine passende Schnittstelle zu ActiveX, DCOM oder CORBA, so daß deren Implementierung einen zusätzlichen Aufwand darstellen würde.

Electronic Data Interchange(EDI) EDI ist ein Standard zum elektronischen, automatisierten Austausch von Geschäftsdaten, wie Bestellungen, Rechnungen oder Lieferscheinen. Dazu werden Nachrichtentypen nach internationalen Normen, wie EDIFACT(Electronic Data Interchange for Administration, Commerce and Transport) oder ANSI X12(Nationaler US-Standard) festgelegt

Die EDI-Realisierung des SAP R/3setzt sich aus den EDI-fähigen Anwendungen, der EDI-Schnittstelle und sogenannten EDI-Subsystemen zusammen. Die EDI-Schnittstelle erlaubt die Formulierung von Regeln für den Empfang oder das Versenden von EDI-Nachrichten, sowie deren Verarbeitung. Die EDI-Nachrichten werden an der Schnittstelle als SAP-spezifische *Intermediate Docmuents*(IDOC) repräsentiert (→[IDO96]). Der Austausch von EDI-Nachrichten mit anderen Systemen läuft über EDI-Subsysteme, die außerdem deren Archivierung und Konvertierung durchführen.

Für die Integration in Tycoon ist EDI nicht geeignet. Zwar wäre die Generierung und Verarbeitung von EDI-Nachrichten in Tycoon-Tl möglich, nur kann damit keine Systemintegration durchgeführt werden. Denn das Ziel der Integration ist eine Leistungserweiterung des Tycoon-Systems um betriebswirtschaftliche Komponenten und nicht die Kommunikation mit dem SAP R/3 auf betriebswirtschaftlicher Ebene.

Application Link Enabling(ALE) Aufgrund geographisch verteilter Unternehmensstrukturen mit gemeinsamen Geschäftsabläufen wird die Kopplung zwischen SAP-Applikationsservern auf einer betriebswirtschaftlichen Ebene zunehmend wichtiger. Diese „lose“Kopplung betriebswirtschaftlicher Module auf verteilten Systemen mit unterschiedlichen Datenbanken wird durch die ALE-Schnittstelle mit einen betriebswirtschaftlich orientierten Nachrichtenaustausch hergestellt (→[BEG96]).

Die ALE-Architektur umfaßt Applikationsdienste, Verteilungsdienste und Kommunikationsdienste (→[ALE96]). Die Applikationsdienste erzeugen und verarbeiten ALE-Nachrichten auf der betriebswirtschaftlichen Abstraktionsebene mit ihren Attributen, Metadaten, wie z.B. der Stammdatenstruktur, und Ereignisse auf Datenebene. Die Verteilungsdienste enthalten logische Verteilungsfunktionen, wie die releaseunabhängige Nachrichtendefinition, Beschreibung der Verteilungsstruktur, Abbilden von logischen auf physische Adressen etc. Die Kommunikationsdienste führen den Nachrichtenaustausch auf Basis des asynchronen RFC (vgl. Abschnitt 4.1.2) oder EDI durch.

Für die Nachrichtenstruktur verwendet ALE *Intermediate Documents*(IDOC). Sie ist vergleichbar mit der EDI-Struktur. Für alle im Anwendungsverbund mehrerer auf verschiedene Applikationsserver mit unterschiedlichen unterliegenden Datenbanken verteilte betriebswirtschaftlichen Module austauschbaren Objekte gibt es einen eigenen IDOC-Typ. Diese Typen lassen sich in Steuerdaten, Stammdaten und Bewegungsdaten klassifizieren.

Ein standardisiertes Verteilungsmodell beschreibt inhaltlich zusammengehörende, verteilbare Funktionstypen. Eine konkretes Modell kann im Rahmen dieser vorgegebenen Szenarien festgelegt werden.

Für die Integration in Tycoon ist ALE aus den gleichen Gründen wie EDI nicht geeignet, obwohl durch die Kommunikationsdienste auch der Nachrichtenaustausch zu R/2- und Fremdsystemen unterstützt wird. Unter der Vorraussetzung der bereits erfolgten Systemintegration wäre die Nutzung der ALE-Schnittstelle zur Kopplung von betriebswirtschaftlichen Tl-Anwendungen mit weiteren SAP-Applikationsservern auf einer betriebswirtschaftlichen Ebene möglich.

Interaktion an der Benutzerschnittstelle

Im Vergleich zu den Interaktionsschnittstellen der Datenbank oder des Applikationsservers, die den direkten Zugriff auf semantisch arme Objekte des SAP R/3 ermöglichen, wie Datenbanktabellen oder Funktionsbausteine, bietet die GUI-Interaktion direkten Zugriff auf SAP-Transaktionen, die einerseits den höchsten Abstraktionsgrad aller Objekte besitzen und andererseits Kommunikationen über das SAPGUI repräsentieren.

Die GUI-Interaktion wird seit SAP R/3 Version 3.0c für algorithmisch gesteuerte Anwender durch die *Intelligent Terminal*-Schnittstelle (→[SAP96a]) realisiert. Sie simuliert in einer Emulation das Dynpro-Protokoll des SAPGUI der konventionellen Benutzerkommunikation und macht die Dynpro-Screens über eine der drei Schnittstellen C-API, Terminal-Service für serielle Anbindungen oder Terminal-Service für Socket-Kommunikation algorithmischen Anwendungen verfügbar. Dazu werden DYNP-Events von dem SAP-Applikationsserver empfangen, zwischengespeichert und wieder versandt. Diese DYNP-Events enthalten vollständige Screen-Beschreibungen, die durch die genannten Schnittstellen den Regeln des SAPGUI entsprechend modifiziert werden können.

Typische Anwendungen dieser Schnittstelle liegen im Bereich der Software-Agenten, Workflows oder Programmen zur Konstruktion weiterer Schnittstellen, wie dem WWW[17], der interaktiven Sprachsteuerung oder den Braille-Terminals für Blinde.

Der Ausblick 6.3 bietet einen Überblick über mögliche Nachfolgearbeiten auf der Basis des Intelligent-Terminals.

3.2 Grundlagen der Programmierung für Funktionsbausteine

Im vorigen Abschnitt wurde der Remote Function Call als die integrative Interaktionsschnittstelle zum SAP R/3-System mit den weitreichensten Möglichkeiten zum Zugriff auf die semantischen Kernkonstrukte des SAP R/3 identifiziert. Die Kommunikationsendpunkte des RFC in einem Applikationsserver sind Funktionsbausteine. Semantisch sind Funktionsbausteine in die Kategorie der ABAP/4-Programme einzuordnen, werden aber in als eigenständige Objekte in der Funktionsbibliothek (vgl. Abschnitt 4) verwaltet.

Im folgenden wird ein Abriß der ABAP/4-Workbench und der Programmierung in der Programmiersprache ABAP/4 als Grundlage der Entwicklung von Funktionsbausteinen gegeben. Im einzelnen wird das Datenmodell vorgestellt, welches in dem Data Dictionary des SAP R/3 abgelegt ist. Dazu gehört eine Beschreibung des hierarchischen monomorphen Typsystems. Dann folgt ein Abschnitt, der einen Überblick der wichtigsten Eigenschaften der Programmiersprache ABAP/4 bietet. Schließlich wird auf dieser Grundlage untersucht, welche sprachlichen Mittel in ABAP/4 existieren, um auf SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen zuzugreifen.

3.2.1 Die Workbench

Die Workbench (Abbildung 3.9) des SAP R/3 enthält als Entwicklungsumgebung alle notwendigen Werkzeuge, um neue Anwendungen zu entwickeln. Alle Entwicklungen bestehen aus *Entwicklungsumgebungsobjekten*, die in *Entwicklungsklassen* zusammengefaßt sind. Entwicklungsumgebungsobjekte können Typen, SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen mit den dazugehörigen

[17] World Wide Web

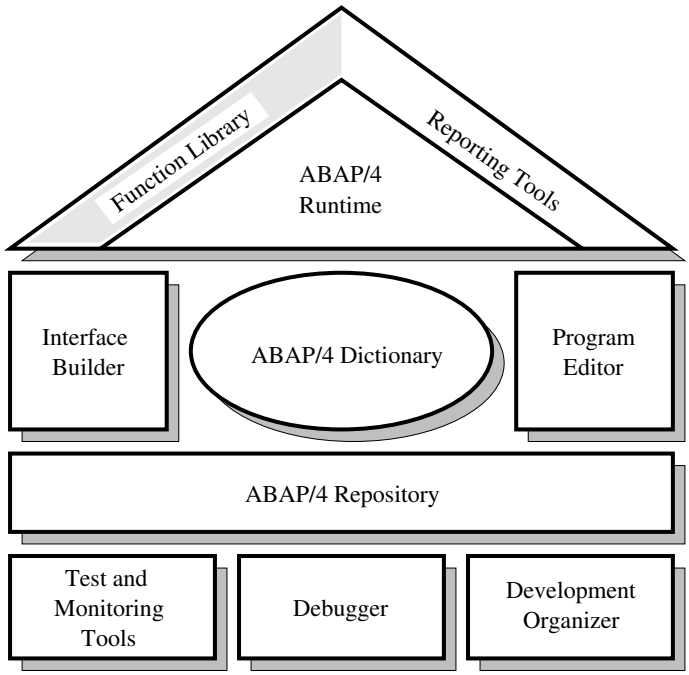


Abbildung 3.9: Die Workbench (→[SAP94a]): Werkzeuge, Verwaltung und Objekte

Screens, Ablauflogiken und Modulen sein (siehe Abschnitt 3.1.2). Die Workbench beherbergt Werkzeuge (→[WBT96a]) zur Herstellung und Änderung aller Arten von Entwicklungsumgebungsobjekten (→[Mat96b]). Durch den Programm-Editor können zum Beispiel ABAP/4-Programme angelegt oder bearbeitet werden. Der Interface-Builder, bestehend aus Screen-Painter und Menu-Painter, erzeugt und verändert Screens respektive Menüs und Drucktastenleisten (vgl. Abschnitt 3.1.2). In [WBT96b] wird eine exemplarische, interaktive Einführung in die Benutzung der Workbench gegeben.

Von besonderer Bedeutung für die vorliegende Arbeit ist die Funktionsbibliothek, die als Teil der Workbench alle Funktionsbausteine verwaltet. Sie wird in Kapitel 4 gesondert vorgestellt.

3.2.2 Das Datenmodell

Als Datenmodell verwendet SAP R/3 das Relationenmodell (→[Dat95, Sch87, Vos95]), welches 1970 von Ed Codd in [Cod70] vorgeschlagen wurde. Bis heute erfuhr es eine tiefgehende mathematische Durchdringung. Auch die praktische Relevanz des Relationenmodells ist außerordentlich hoch, weil die Struktur sehr vieler realweltlicher Daten genau mit der Metamodellierung des Relationenmodells übereinstimmt. Der folgende kurze Überblick hält sich bei der Wahl der Begriffe an [Dat95].

Wesentliches Merkmal des Relationenmodells sind Tabellen. Sie aggregieren paarweise verschiedene Tupel. Tupel wiederum aggregieren Attribute. Attribute sind Kombinationen aus einem Namen und einer Domäne oder einem Typ, wie String, Zeit oder Ganzzahl. Die Zahl der Attribute einer Tabelle heißt ihr Grad und die Zahl der Tupel ist ihre Kardinalität.

Zu Tabellen gibt es Schlüsselkandidaten. Das sind Attributkombination mit zwei Eigenschaften. Ersten müssen die Attribute alle Tupel der Tabelle eindeutig identifizieren, das heißt daß die Tupel funktional abhängig vom Schlüsselkandidaten sind. Zweitens darf die Attributkombination des Schlüsselkandidaten nicht reduzierbar sein, so daß bereits eine echte Teilmenge der Schlüsselattribute jedes Tupel der Tabelle eindeutig identifiziert. Da keine zwei Tupel einer Tabelle gleich sind, gibt es wenigstens einen Schlüsselkandidaten für jede Tabelle. Andererseits kann eine Tabelle mehrere Schlüsselkandidaten besitzen.

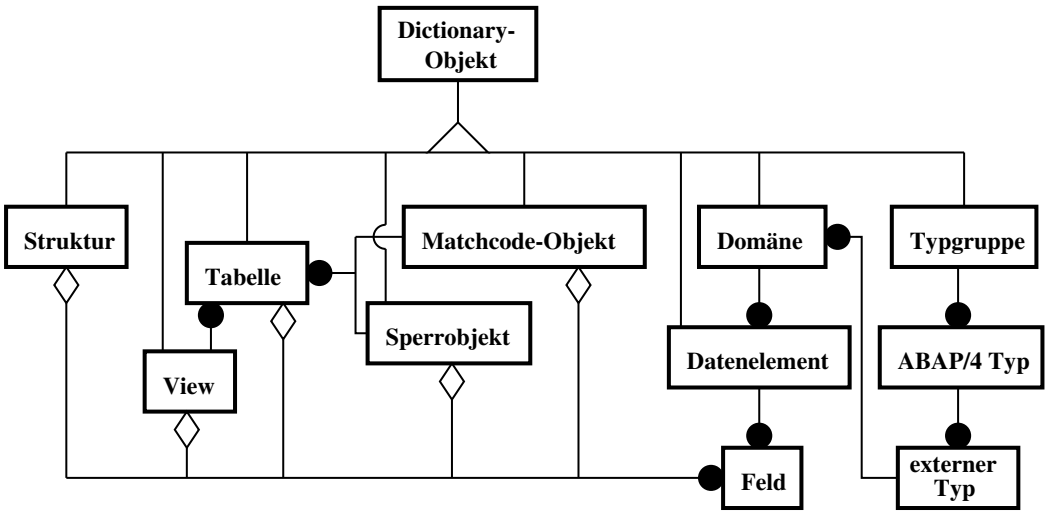


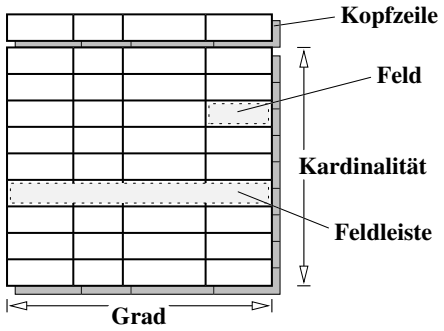
Abbildung 3.10: Das SAP Data Dictionary

Wichtige Schlüssel im SAP-System sind Primärschlüssel und Fremdschlüssel. Unter allen Schlüsselkandidaten kann einer als der Primärschlüssel ausgewählt werden. Die übrigen werden dann als alternative Schlüsselkandidaten bezeichnet.

Fremdschlüssel treten im Kontext der Verkettung mehrerer Tabellen auf. Angenommen es liegen zwei Tabellen T_1 und T_2 vor. Wenn T_1 einen Schlüsselkandidaten enthält, der als Primärschlüssel für T_2 fungiert, dann heißt er Fremdschlüssel in T_1. Häufig wird als Integritätsbedingung gefordert, daß T_1 nur solche Tupel enthalten darf, deren Fremdschlüsselattribute eine Entsprechung in T_2 finden. Diese Bedingung heißt *referentielle Integrität*.

Im Kontext des SAP R/3 finden alle wichtigen Eigenschaften des Relationenmodells eine Anwendung. Allerdings wird ein monomorphes, mehrschichtiges Typsystem genutzt (siehe Abschnitt 3.2.2). Die wichtigsten Begriffe des Relationenmodells mit ihrer SAP-Entsprechung sind in der nebenstehenden Tabelle aufgeführt.

Zur *Kopfleiste* gibt es keine direkte Entsprechung im Relationenmodell. Der Grund dafür ist die Darstellung der Tabellen als mengenwertige Variablen in einem realen System. Es werden Tupel gelöscht, hinzugefügt, inspiziert und modifiziert. Dies wird häufig über eine Cursor-Schnittstelle getan, die einen Speicherbereich für ein Tupel verwaltet. Dazu addressiert man das betreffende Tupel, welches darauf im Cursor erscheint und mutiert werden kann. Die Bezeichnung für den Cursor in SAP ist Kopfleiste.



Der Datenkatalog

Der Datenkatalog des SAP R/3, dem *Data Dictionary*, dient der zentralen Verwaltung aller Metadaten des Systems. Nach [DDI96] sind seine Aufgaben:

- Verwaltung von Informationen.
- Bereitstellung von Informationen für Auswertungen.
- Unterstützung der Softwareentwicklung.

- Unterstützung der Dokumentation.
- Sicherung von Flexibilität und Aktualität von Datendefinitionen.

Zusammengefaßt speichert und verwaltet das Data Dictionary des SAP R/3 Informationen über Datendefinitionen und macht sie nach Art eines aktiven Data Dictionary (→[Reu87]) interaktiv zugänglich, so daß ein Entwickler sich vollständig informieren bzw. Definitionen direkt aus dem Data Dictionary in seine Entwicklungen übernehmen kann. Jedes Data Dictionary-Objekt ist mit einem kurzen Kommentar versehen, wodurch seine Bedeutung beschrieben wird. Außerdem gibt es vielfältige Vernetzungen innerhalb des Data Dictionary, was zu der Verständlichkeit der Definitionen beiträgt, weil der jeweilige Datenkontext hergestellt wird.

Über das *Repository Information System* kann neben dem Data Dictionary sogar das gesamte Repository des SAP R/3 Systems inspiziert werden. Es unterstützt das gezielte Suchen nach Repository-Objekten mit bestimmten Eigenschaften. Die Eigenschaften können mittels einfacher logischer Verknüpfungsoperatoren spezifiziert werden. Als Ergebnis erhält man eine Trefferliste der gefundenen Objekte und kann daraufhin interaktiv Objekte auswählen.

Die Objekte erster Klasse des Data Dictionary, in ihren Beziehungen als Objektmodell in Abbildung 3.10 dargestellt, sind folgende:

Strukturen sind Feldkombinationen. Sie können zur Definition von Tabellen übernommen werden oder aber innerhalb von ABAP/4 für die Deklaration von Feldleisten genutzt werden.

Tabellen entsprechen direkt Datenbanktabellen. Ihre Definition umfaßt mehrere Felder mit ihrem Namen und Typ. Der Typ eines Feldes wird durch die Angabe eines Datenelements festgelegt. Außerdem wird ein Primärschlüssel definiert, indem alle ihm zugeordneten Felder markiert werden. Desweiteren werden Fremdschlüsselverbindungen zur sogenannten *Prüftabelle* festgelegt (siehe dazu 3.2.2).

Datenelemente stellen betriebswirtschaftlich orientierte Typen der Modellierung dar. Sie basieren auf Domänen. Bei ihrer Definition können bis zu drei Feldbezeichner unterschiedlicher Länge und eine Überschrift angegeben werden.

Domänen sind technisch orientierte Typen. Sie beinhalten Angaben über den zugrundeliegenden externen Typ, die Ausgabelänge und die Ausgabeformatierung.

Views aggregieren Felder mehrerer verschiedener Tabellen. Sie unterstützen eine zusammengefaßte Sicht auf diese Tabellen. Dabei werden die unterschiedlichen View-Typen Datenbank-, Customizing-, Streich-, Help- und Entitätsview unterstützt.

Matchcodes realisieren sekundäre Zugriffspfade. Das sind Zugriffsmöglichkeiten über Feldkombinationen, die nicht dem Primärschlüssel der Tabelle angehören. Sie können in Dynpros als Eingabehilfe eingesetzt werden.

Sperrobjekte dienen dem gegenseitigen Ausschluß konkurrierender Zugriffe. Sie aggregieren die Tabellen und die betroffenen Felder. Allerdings bilden Sperrobjekte keine vom System automatisch kontrollierten Einheiten. Vielmehr werden bei ihrer Erzeugung Funktionsbausteine generiert, die explizit in ABAP/4-Programmen aufgerufen werden müssen, um einen gegenseitigen Ausschluß beim Zugriff auf bestimmte Tabellenbereiche zu gewährleisten. Da ohne weiteres Programme geschrieben werden können, die diesen Mechanismus misachten, handelt es sich bei diesen Sperrobjekten um eine Form der expliziten Synchronisation.

Typgruppen enthalten Typdeklarationen, die in der Programmiersprache ABAP/4 Verwendung finden. Auf diese Weise werden benutzerdefinierbare Datentypen der Sprachebene angelegt, die zentral im Data Dictionary verwaltet werden. Sie sind zwar Teil des Data Dictonary, haben aber keinen direkten Bezug zu anderen Data Dictionary Objekten.

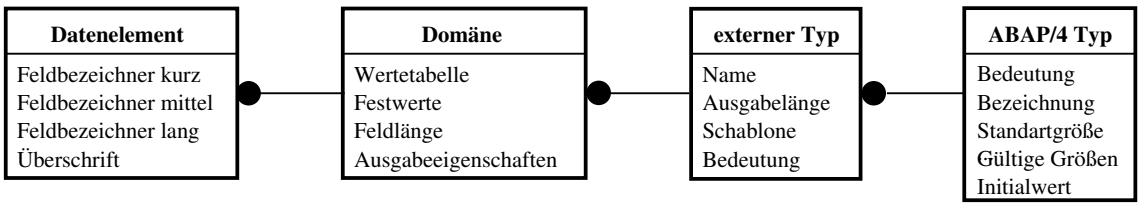


Abbildung 3.11: Die SAP R/3-Typhierarchie

Speziell die Datenelemente und Domänen des Data Dictionary sind Bestandteil des SAP R/3-Typsystems. Sie werden deshalb im folgenden Abschnitt, der eine detaillierte Beschreibung der Typ- und Modellierungshierarchie für atomare Daten enthält, wieder aufgeführt. Die Typgruppen stehen zwar ebenfalls im Zusammenhang mit Modellierung atomarer Typen, jedoch ausschließlich auf dem Niveau der ABAP/4-Sprachebene. Sie enthalten nicht die semantische Bereicherung der Datenelemente und Domänen. Daher spielen sie für die Modellierung eine untergeordnete Rolle.

Das Typsystem

Das im Data Dictionary abgelegte Datentypsystem liegt der kompletten Modellierung im SAP R/3 zugrunde. Wie in Abbildung 3.11 dargestellt, ist das Datentypsystem des SAP R/3 Systems mehrschichtig. Grob kann zwischen Darstellungstypen, programmiersprachliche Typen und Typen der Datenbank unterschieden werden. Wobei letztere vollständig durch die Datenbankschnittstelle verborgen werden.

Die Darstellungstypen sind Datenelemente, Domänen und externe Typen. Datenelemente basieren auf Domänen, welche ihrerseits durch externe Typen erklärt werden. Datenelemente sollen die betriebswirtschaftlichen Eigenschaften einer Typmodellierung beschreiben. Dazu enthalten sie außer drei Feldbezeichnern zur Benennung eine Überschrift. Diese Angaben werden bei der Präsentation von Daten dieses Datenelements auf einem Dynpro genutzt.

Domänen werden durch die Angabe eines externen Typs definiert und mit verschiedenen technischen Attributen erweitert. Für solche externe Typen die keine Länge vorgeben, muß diese in der Domäne festgelegt werden. Das betrifft zum Beispiel den externen Typ `CHAR`, der für Längen > 1 Strings darstellt. Man kann den Wertebereich einer Domäne aus dem externen Typ übernehmen oder ihn durch eine Aufzählung der zugelassenen Werte direkt angeben. Auch die Festlegung einer Tabelle als Wertetabelle für die Domäne ist zulässig. Desweiteren können die Ausgabeeigenschaften beeinflußt werden. So kann die Ausgabelänge vorgegeben werden oder auch die Anzahl der Dezimalstellen hinter einem Komma für den Typ `DEC`.

Von den externen Typen gibt es, wie in Abbildung 3.12 aufgeführt, genau neunzehn. Sie haben sehr heterogene Eigenschaften. Einige der externen Typen haben bereits betriebswirtschaftlichen Charakter, während andere algebraisch oder technisch orientiert sind. Zum Beispiel gibt es den externen Typ `CLNT`, der technisch gesehen drei Byte belegt und speziell für mandantenabhängige Tabellen dem Typ des Mandantenfeldes zugrundeliegt. Andererseits gibt es den externen Typ `FLTP`, der Fließkommazahlen repräsentiert. `NUMC` stellt einen numerischen Text dar, der nicht für Berechnungen vorgesehen ist aber Zahlen beliebiger Länge aufnehmen kann und `INT4` ist eine durch vier Byte kodierte ganze Zahl.

Die programmiersprachlichen Typen, die in der Sprache ABAP/4 verfügbar sind, sind in Abbildung 3.13 aufgeführt. Sie können teilweise durch eine Längenangabe ergänzt werden. So wird aus einem Zeichen mit der Typbezeichnug `C` dann ein String der Länge fünf mit der Typbezeichnung C(5). Allerdings gilt das nicht für alle ABAP/4 Typen. Zum Beispiel hat der Typ Zeit mit dem Typbezeichner `T` immer die vorgegebene Länge sechs.

Die Abbildung der externen Typen auf die ABAP/4-Typen wird im SAP-System durch den Dialog-Prozessor geleistet, so daß eine Datenmodellierung unabhängig von programmiersprachlichen Konstrukten auf Basis der Darstellungstypen in dem Data Dictionary vorgenommen werden kann und nachfolgend Programme erstellt werden können, die über ABAP/4-Typen auf die definierten Dictionary-

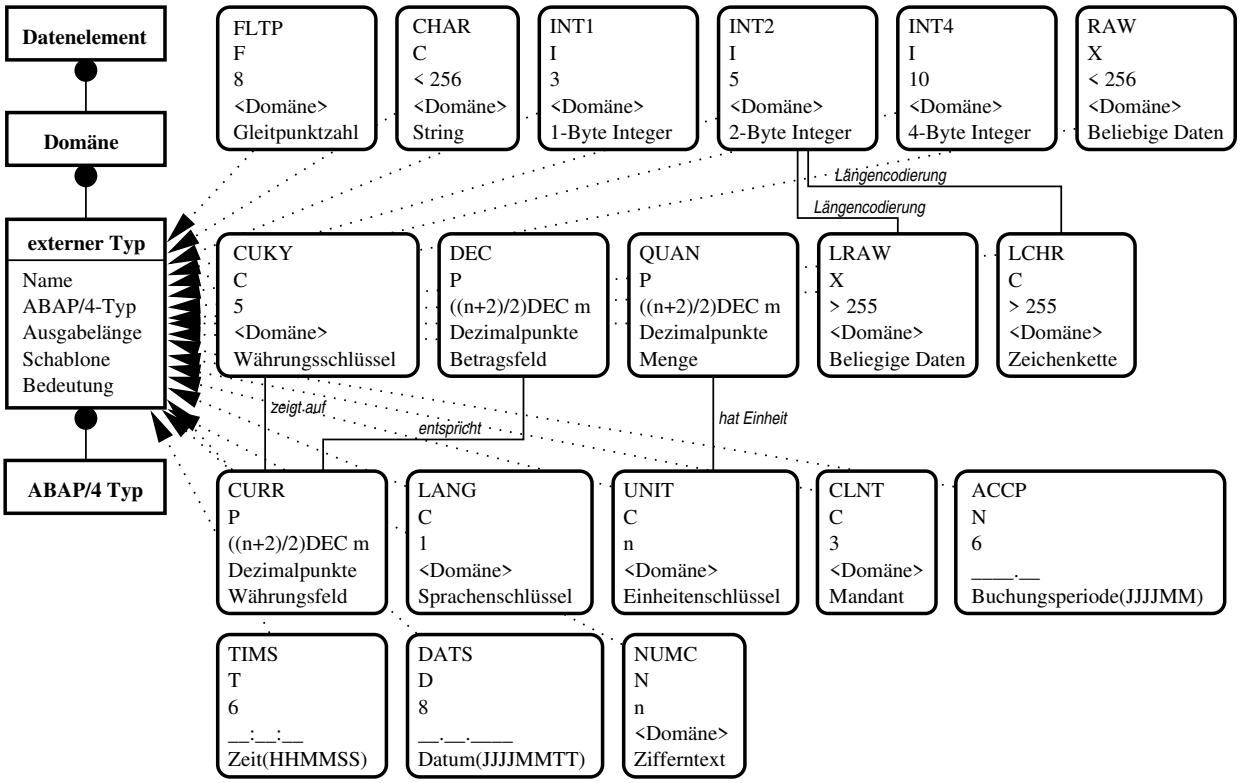


Abbildung 3.12: Menge der externen Typen

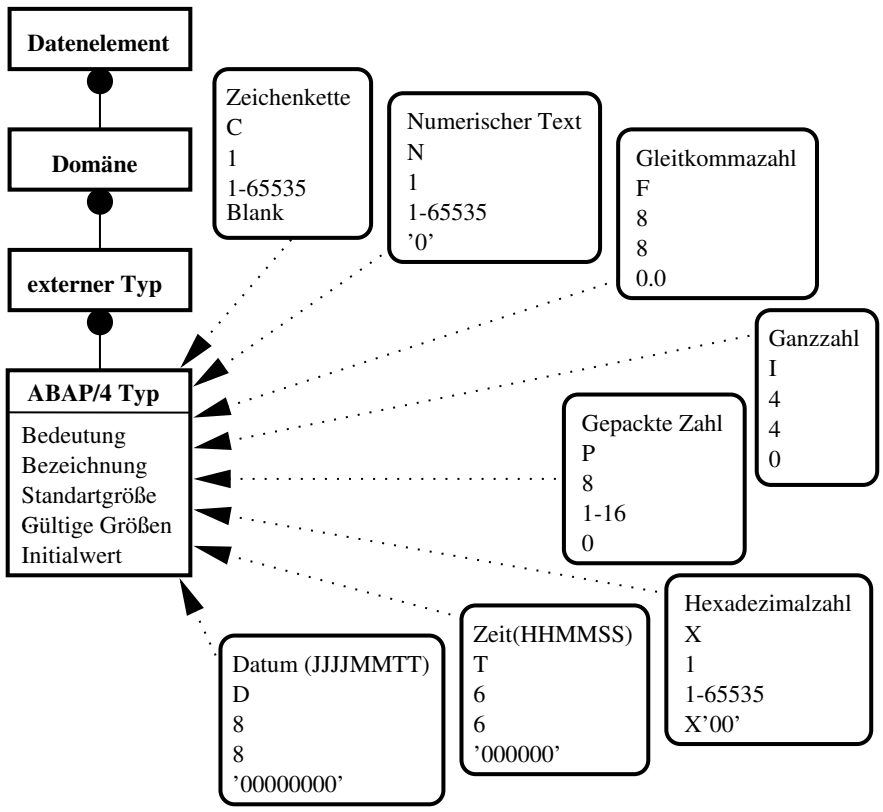


Abbildung 3.13: Menge der ABAP/4 Typen

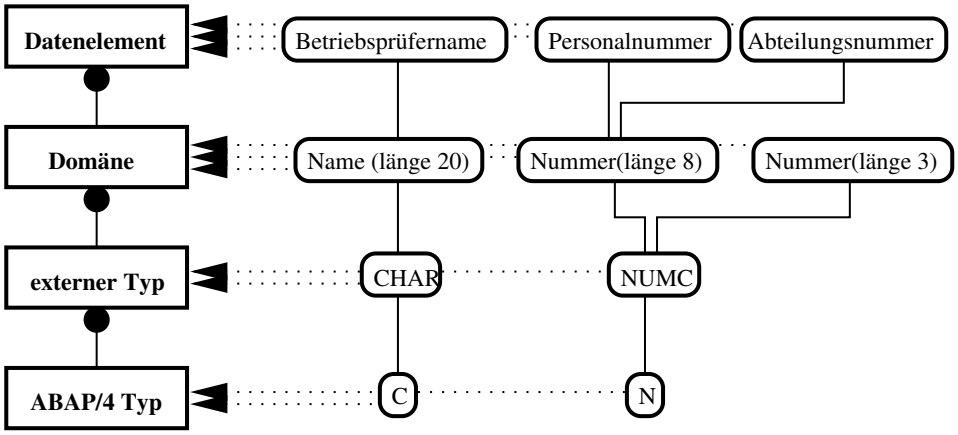


Abbildung 3.14: Zusammenhang der verschiedenen Datentypebenen des SAP R/3

Objekte zugreifen können. Den Rückschluß darauf, welcher ABAP/4-Typ einem konkreten externen Typ zugrunde liegt, muß der Programmierer tun.

Zwischen den ABAP/4 Typen gibt es automatische Typumwandlungen für alle Kombinationen ausgenommen der Typen `D` und `T`, die sich gegenseitig nicht automatisch ineinander konvertieren lassen. Eine zusätzliche Einschränkung liegt darin, daß meistens von Werten des Quelltyps einer Umwandlung gefordert wird, bestimmte Vorbedingungen zu erfüllen. Will man z.B. ein Zeichenfeld vom Typ `C` in einen Wert vom Typ `T` konvertieren, dann muß das Zeichenfeld mit einer sechsstelligen Zahl der Semantik HHMMSS[18] beginnen. Ansonsten kann es zu Laufzeitfehlern kommen.

Ein Beispiel für die Verwendung der vier Typebenen ABAP/4-Typ, externer Typ, Domäne und Datenelement ist in Abbildung 3.14 wiedergegeben. Ein Betriebsprüfername wird demnach als ein entsprechendes Datenelement angelegt. Das basiert auf einer Domäne, die die technischen Eigenschaften des Namens, insbesondere seine Speicherlänge, beschreibt. Der Domäne ihrerseits liegt der externe Typen `CHAR` zugrunde.

3.2.3 Die Programmiersprache ABAP/4

Ursprünglich als einfache Sprache zur Drucklistenaufbereitung im System SAP R/2 gedacht, was sich nicht zuletzt in der früheren Bezeichnung „**A**llgemeiner **B**erichtsauf**a**rbeitungs**p**rozeß ausdrückt (→[Mat96b]), wurde anfang der achziger Jahre ABAP dann nach und nach zu einem vollständigen Entwicklungswerkzeug für betriebswirtschaftliche Anwendungen ausgebaut und die Bezeichnung **A**dvanced **B**usiness **A**pplication **P**rogramming Language mit dem Zusatz **4** für eine Sprache der vierten Generation eingeführt, kurz ABAP/4.

ABAP/4 ist integraler Bestandteil der R/3-Workbench. Ein ABAP/4-Programm besitzt nicht die Eigenständigkeit, wie C, Pascal oder Cobol, deren Quelltexte mit beliebigen Editoren erstellt, dann mit einem Compiler in die Sprache der Wirtsmaschine übersetzt und schließlich durch das Betriebssystem gestartet werden. ABAP/4-Programme werden direkt durch den Workbench-Editor angelegt und in dem SAP-Repository gespeichert. Sie müssen dann generiert und aktiviert werden, wodurch eine SAP-interne Repräsentation erzeugt wird, die dann durch einen Interpreter der Art einer virtuellen Maschine ausgeführt werden kann. Die möglichen Zustände eines ABAP/4-Programms sind vergleichbar mit denen von Tl- oder C-Programmen (siehe Tabelle 3.1): Durch diese Leistung der SAP-Middleware, ABAP/4-Programme zu erzeugen, zu verwalten und auszuführen, werden alle ABAP/4Programme vollständig portabel[19].

[18] HH=Stunde, MM=Minute, SS=Sekunde

[19] Durch die Native-SQL Schnittstelle können allerdings datenbankherstellerabhängige ABAP/4-Programme geschrieben werden

ABAP/4	Tl	C
Quelltext	Quelltext	Qelltext
generiert	übersetzt	übersetzt
aktiviert	gebunden	gebunden

Tabelle 3.1: Zustände eines ABAP/4-Programms im Vergleich

ABAP/4 wird laufend weiterentwickelt. Durch das Hinzufügen neuer Schlüsselwörter paßt SAP die Sprache an neue Anforderungen an. Zum Beispiel ist ab Version 3.0 die Verwendung von Typdeklarationen durch das Schlüsselwort `TYPE-POOLS` eingeführt worden. In ähnlicher Weise vollzieht sich der Ausbau anderer Bereiche von ABAP/4. Gelegentlich werden aber auch veraltete Schlüsselworte mit dem Vermerk versehen, daß sie zukünftig eventuell nicht mehr unterstützt werden.[20] Außerdem gibt es in ABAP/4 Befehle, die sehr stark spezialisierte Leistungen erbringen, wie z.B. die Open-SQL-Befehle[21]. Dies sind typische Eigenschaften von 4GL-Sprachen, deren Syntax und semantische Objekte sich an der Anwendungsdomäne orientieren.

Es gibt drei Regeln für die syntaktische Gestaltung von ABAP/4-Code (→[ABB96]):

1. Schlüsselworte leiten neue Kommandos ein.

2. Nach jedem Kommando kommt ein Punkt.

3. Wenn mehrere Kommandos mit dem gleichen Präfix aufeinanderfolgen, kann eine abkürzende Schreibweise benutzt werden, in der der Präfix einmalig vorangestellt und durch einen Doppekpunkt beendet wird, gefolgt von der Komma-separierten Liste aller Kommandoendungen mit einem abschließenden Punkt.

Die Liste der Schlüsselworte ist sehr umfangreich und wegen der oben genannten Gründe nur in der Online-Dokumentation[22] vollständig zu finden. Jedoch ist nach [SAP94a] eine Kategoriesierung der ABAP/4-Sprachelemente folgendermaßen möglich:

Deklarative Sprachelemente beschreiben den Aufbau der Daten, die in einem Programm verwendet werden. Hierzu gehören Anweisungen, wie `TABLES`, `TYPE-POOLS`, `TYPES` und `DATA` . Dabei sind `TABLES` und `TYPE-POOLS`, die sich direkt auf Data Dictonary-Tabellen und -Typen Pools beziehen graduell verschieden von `TYPES` und `DATA`, die ausschließlich ABAP/4-Typen deklarieren bzw. definieren.

Operationale Sprachelemente ermöglichen Datenmanipulationen. Das sind neben Berechnungen auch Zuweisungen. Wichtige Schlüsselwörter dieser Kategorie sind unter anderen `ADD`,`SUBSTRACT`, `DIVIDE`, `MULTIPLY`, `MOVE`, `COMPUTE`.

Steuerungssprachelemente realisieren die aus Sprachen der dritten Generation bekannten Kontrollstrukturen, wie Schleifen Verzweigungen und Unterprogramme. Dazu gehören unter anderem die Schlüsselworte `DO`,`IF` und `PERFORM`.

Zeitpunkt-Sprachelemente verknüpfen Programmteile mit bestimmten Ereignissen, die bei der Abarbeitung eintreten können. Diese Seiteneffekte sind entweder direkt mit der Verarbeitung zusammenhängende Ereignisse, wie z.B. der Start einer neuen Seite bei der Erzeugung einer Reportliste, oder Benutzereingaben. Beispiele für Schlüsselworte dieser Kategorie sind `TOP-OF-PAGE` und `AT USER-COMMAND`.

[20] Die Entfernung von solchen alten Sprachelementen ist mit dem Risiko verbunden, daß die Abwärtskompatibilität nicht mehr gewährleistet ist, so daß ABAP/4 mit jeder Version umfangreicher und unübersichtlicher wird.

[21] 3GL-Sprachen bieten solche Leistungen entweder durch Zusatzbibliotheken oder durch Syntaxeinbettungen an

[22] [SAP96a]

Die ersten drei Kategorien von Sprachelementen unterstützen einen imperativen Programmierstil. Dabei werden Typen definiert, Variablen, Konstanten und Datenbanktabellen deklariert. Durch operationale Sprachelemente können seiteneffektbasiert Änderungen auf Variablen und Datenbanktabellen vorgenommen werden. Steuerungssprachelemente regeln den Kontrollfluß, wie die bedingte Verzweigung oder einen Unterprogrammaufruf. Das dynamische Verhalten eines ABAP/4-Programms ergibt sich damit aus der Beschreibung, die durch den Quelltext vorgegeben ist.

Hingegen dient die vierte Kategorie der Zeitpunkt-Sprachelemente einer externen Steuerung des Programmflußes. Dabei setzt das SAP-System im Zusammenhang mit Ereignissen, die entweder durch andere ABAP/4-Programme oder durch Benutzereingaben angestoßen werden, den Programmkontrollfluß hinter solchen Zeitpunktsprachelementen fort. Zum Beispiel wird bei der Ausgabe eines Reports automatisch die Anweisung `TOP-OF-PAGE` angesprungen, wenn eine Seite beginnt. Dadurch, daß die Zeitpunkt-Sprachelemente zur Unterstützung spezieller Ereignisse der SAP-Systemumgebung integraler Teil der Sprache ABAP/4 sind, wird der hohe Grad der Integration und gegenseitigen Abhängigkeit zwischen ABAP/4 und dem SAP-System deutlich.

Strukturen, Feldleisten und interne Tabellen

Während der Abarbeitung von ABAP/4-Programmen werden häufig mengenwertige Datentypen zur Aufnahme komplexer, umfangreicher Zwischenergebnisse gebraucht. Diesen Zweck erfüllen Feldleisten und interne Tabellen (→[ABB96]). Sie sind keine Data Dictionary oder Repository-Objekte und sind deshalb nur zur Laufzeit verfügbar. Bei Beendigung eines Programms verlieren sie ihren Inhalt.

Feldleisten sind Kombinationen aus mehreren Feldern. Der Typ einer Feldleiste ist eine Struktur, die entweder in ABAP/4 oder durch einen Verweis auf eine entsprechende Data Dictionary-Struktur deklariert wird. Strukturen sind aber Sequenzen aus Typen oder wiederum Strukturen, wobei Rekursionen nicht zugelassen sind. Deshalb kann durch ein „Plattklopfen“ zu jeder beliebig verschachtelten Struktur die zugehörige Sequenz der ABAP/4-Typen gebildet werden, die bereits die vollständige Typbeschreibung einer Feldleiste enthält. Feldleisten sind die einzigen strukturierten Datenobjekte in ABAP/4.

Interne Tabellen ähneln Feldleisten, bestehen aber, wie die Datenbanktabellen (siehe Abschnitt 3.2.2), aus mehreren Feldleisten und eventuell einer Kopfleiste. Existiert die Kopfleiste, dann kann sie äquivalent zu Feldleisten behandelt werden. Der von internen Tabellen benötigte Speicherplatz wird statisch mit einer vorgegebenen Größe im Rollbereich[23] initialisiert und zur Laufzeit dynamisch angepaßt, indem entsprechende Bereiche in den Paging-Bereich ausgelagert werden. Sie werden in ABAP/4 deklariert und definiert und beziehen, wie die Feldleisten die Struktur ihrer Felder daher aus ABAP/4-Typen im Unterschied zu den transparenten Datenbanktabellen, deren Feldtypen durch die Angabe von Datenelementen angegeben wird.

Folgender ABAP/4 Quellcode deklariert eine Struktur `Person`, eine interne Tabelle `PersonTable` und definiert dann die interne Tabelle `Persons` mit 55 voralloziierten Zeilen im Rollbereich. Zusätzlich wird die Tabelle `PersonsKL` durch das Schlüsselwort `WITH HEADER LINE`, wie `Persons` mit einer Kopfleiste definiert.

```
TYPES:  BEGIN OF Person,
          NAME(20) TYPE C,
          AGE      TYPE I,
        END OF Person.
TYPES: PersonTable TYPE Person OCCURS 55.
DATA: Persons TYPE PersonTable,
      PersonsKL LIKE PersonTable WITH HEADER LINE.
```

[23] Die Roll- und Paging-Bereiche sind Teil der Speicherverwaltung in SAP R/3 (→[MME96]).

Zur Modifikation und Inspektion interner Tabellen stehen unter anderem[24] folgende Operationen zur Verfügung:

`APPEND` Anfügen einer neuen Feldleiste

`DELETE` Löschen einer Feldleiste

`INSERT` Einfügen einer neuen Feldleiste

`MODIFY` Überschreiben einer Feldleiste

`LOOP` über alle Feldleisten der Tabelle iterieren

`READ` explizites Lesen einer Feldleiste einer internen Tabelle

`SORT` Sortieren der internen Tabelle

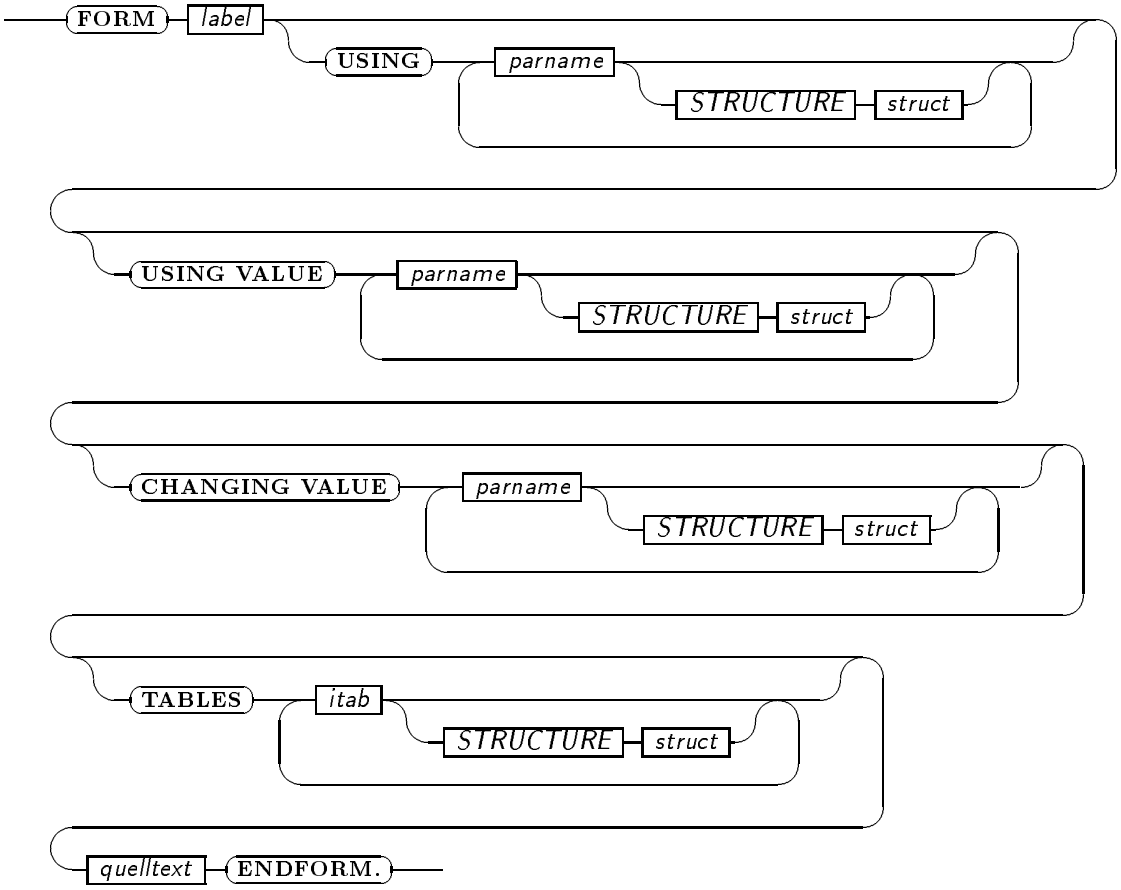
`COLLECT` Anfügen einer neuen Feldleiste oder, bei Übereinstimmung des Schlüssels mit einer vorhandenen Feldleiste, Aufsummieren der numerischen Felder

Die Selektion der Feldleiste geschieht bei dem Zugriff auf interne Tabellen durch eine explizite Angabe seiner Nummer mittels `INDEX <number>` oder durch die Angabe einer Schlüsselkombination über `WITH KEY <key>=<value>*`. Dabei sind die nichtnumerischen Felder einer internen Tabelle als Schlüsselkombination voreingestellt.

Lokale ABAP/4-Unterprogramme

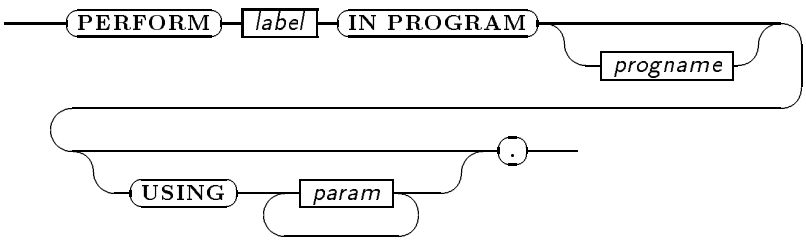
Die Strukturierung von ABAP/4-Programmen kann neben der Möglichkeit der Aufsplittung in mehrere ABAP/4-Programme (vgl. Abschnitt 3.2.4) oder Funktionsbausteine (vgl. Kapitel 4) durch lokale Unterprogramme erfolgen:

[24] Weitere Operationen sind anwendbar(→[SAP96a]): `DESCRIBE`, `FREE`, `REFRESH`, `CLEAR`



Es sind für die Unterprogramme die drei Parametervarianten „call by reference“, „call by value“und sogenannte „changing“-Parameter möglich, die sich wie „call by value“-Parameter verhalten bis die `ENDFORM`-Anweisung durchlaufen wird, die eine Rückübertragung geänderter Werte veranlaßt.

Der Aufruf von lokalen Unterprogrammen wird mit



durchgeführt. Mit `progname` wird optional das ABAP/4-Programm angegeben, das das Unterprogramm enthält. Die Parameter werden in der angegebenen Reihenfolge an die Formalparameter des Unterprogramms gebunden.

3.2.4 Programmierung mit semantischen Objekten des SAP R/3

Im folgenden werden die Mechanismen der Sprache ABAP/4 zur Programmierung mit den semantischen Objekten SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen aufgezeigt. Sie liefern die Begründung für die Offenheit der RFC-Schnittstelle.

Datenbankzugriff auf SAP-Tabellen

Wie bereits in 3.1.1 erwähnt wird der Datenbankzugriff in SAP durch die in ABAP/4 eingebetteten SQL-Dialekte *Open-SQL* und *Native-SQL* realisiert. Native-SQL spielt allerdings eine untergeordnete Rolle, da es nur eine Einbettung des SQL-Dialekts der unterliegenden Datenbank in ABAP/4 darstellt und somit herstellerabhängig und nicht portabel ist. Zudem können bei der Verwendung von Native-SQL Inkonsistenzen in dem Datenmodell von R/3 auftreten, da die Datenbankschnittstellenabbildung der ABAP/4-Typen auf die Datenbanktypen umgangen wird und desweiteren eventuelle Abhängigkeiten zwischen den Tabellen des R/3 nicht respektiert werden.

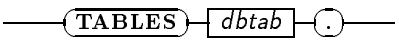
Die Verwendung von Native-SQL mit Datenübertragun durch Hostvariablen läßt sich durch ein Beispiel verdeutlichen:

```
DATA: BEGIN OF PERSON,
      NAME(20) TYPE C,
      AGE TYPE I.
DATA: END OF PERSON.
. . .
EXEC SQL.
  SELECT * FROM PERSONDB
    WHERE NAME = "Heinz"
      INTO :PERSON ENDEXEC.
```

Hier wird eine Zeile der Datenbank `PERSONDB` in die Feldleiste `PERSON` eingelesen. Wenn durch die Bedingungen der `WHERE`-Klausel mehrere Zeilen der Datenbank selektiert werden, dann kann statt `INTO` mittels des Schlüsselwortes `PERFORM` ein ABAP/4-Unterprogramm angegeben werden, das für jede der selektierten Zeilen genau einmal aufgerufen wird.

Der präferierte Zugriff auf Datenbanktabellen sollte die Open-SQL-Schnittstelle verwenden. Open-SQL ist unabhängig von der eingesetzten Datenbank auf allen Plattformen mit einem einheitlichen vollständig in ABAP/4 integrierten Sprachumfang verfügbar. Die Datenbankschnittstelle übernimmt die Umsetzung der Open-SQL Befehle in den SQL-Dialekt der jeweiligen Datenbank.

Open-SQL umfaßt ausschließlich DML[25]-Befehle. Die DDL[26] zur Definition der Tabellen ist ein Teil des Data Dictionary (siehe Abschnitt 3.2.2). Bei Verwendung von Tabellen in ABAP/4 muß lediglich eine Deklaration vorgenommen werden:



Der Befehlssatz für Open-SQL umfaßt die üblichen SQL-DML-Schlüsselworte, eine Cursor-Schnittstelle und Befehle zum Beenden oder Zurücksetzen flacher Transaktionen(SAP-LUW) (vgl. Abschnitte 3.1.2 und 3.2.5):

`SELECT` Abfragen von Datensätzen nach vorgegebenen Bedingungen

`INSERT` Einfügen eines oder mehrer Datensätze eventuell aus einer internen Tabelle

`UPDATE` Ändern eines Datensatzes.

`MODIFY` Einfügen oder Ändern von Datensätzen. Dieses Kommando ist eine Kombination aus INSERT und UPDATE.

`DELETE` Löschen von Datensätzen

`OPEN CURSOR` Erzeugen eines Cursors

[25]Data Manipulation Language
[26]Data Definition Language

`FETCH NEXT CURSOR` Einlesen des nächsten Datensatzes in den Cursor

`CLOSE CURSOR` Zerstören des Cursors

`COMMIT WORK` Beenden der aktuellen SAP-LUW

`ROLLBACK WORK` Zurücksetzen der aktuellen SAP-LUW

Dynamischer Aufruf von ABAP/4-Programm

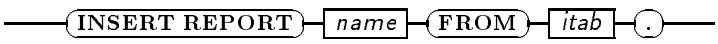
Die dynamische Programmsteuerung (→[WAR96]) erlaubt es, ABAP/4-Programme zur Laufzeit zu erzeugen, zu inspizieren oder auszuführen. Dadurch wird ein reflektiver Zugriff auf ABAP/4-Programme ermöglicht und somit können Algorithmen, die erst zur Laufzeit bekannt sind, in den Programmablauf eingebaut werden.

Die ABAP/4-Anweisung zur dynamischen Ausführung von ABAP/4-Programmen ist:

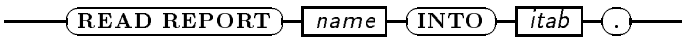
Sie wird durch Anweisungen zum dynamischen Zugriff auf ABAP/4-Programme ergänzt, die eine Laufzeitrepräsentation des Programmtexts in einer internen Tabelle vom Typ `C` und der Länge 72 annehmen, wie zum Beispiel:

```
DATA CODE(72) TYPE C.
```

Die Anweisung zum Einlesen eines Programms in eine interne Tabelle ist:



Mit



wird eine neues Programm erzeugt.

Die dynamische Programmsteuerung ermöglicht es ABAP/4-Unterprogramme dynamisch zu generieren, die Open SQL-Anweisungen enthalten. Kombiniert man dies mit der Möglichkeit Open-SQL-Ergebnisse als interne Tabellen zu erhalten, dann resultiert das in dynamisch ausgeführten SQL-Anweisungen. Andere Anwendungen dieser Art der Programmsteuerung liegen z.B. in der Fernsteuerung algorithmischer Abläufe und laufzeitreflektiver Berechnungen im SAP R/3.

Ausführen von SAP-Transaktionen

Durch das *Batch Input*-Verfahren können SAP-Transaktionen programmgesteuert ablaufen. Der eigentlich Zweck des Batch Inputs ist die automatische Übernahme von Massendaten aus externen Systemen über SAP-Transaktionen, die den Vorteil bieten, wirklich alle Konsistenzbedingungen der betriebswirtschaftlichen Anwendungen zu garantieren. Die Vorgehensweise sieht dabei eine Konvertierung und Übernahme externer Daten in SAP-Tabellen vor, von wo sie noch in eine spezielle Struktur gewandelt werden müssen, der der Batch Input die Eingabedaten für die Dynpros einer SAP-Transaktion entnimmt. Die Ausführung geschieht dann entweder indirekt über die Hintergrundsteuerung oder durch einen direkten Aufruf der SAP-Transaktionen mit entsprechenden ABAP/4-Befehlen. Die Struktur der Eingabedaten, die als interne Tabelle übergeben werden, muß der in Abbildung 3.15 entsprechen. Die Felder der Tabelle lassen sich in zwei Gruppen aufteilen, deren Belegung mit Werten sich gegenseitig ausschließt. Die erste Gruppe mit `PROGRAM`, `DYNPRO` und `DYNBEGIN` addressiert ein neues Dynpro zur Verarbeitung. Die zweite Gruppe, bestehend aus `FNAM` und `FVAL`, nimmt Paare aus einem Feldnamen und einem Feldwert auf. Die Tabelle wird nun derart mit Zeilen gefüllt, daß immer eine Zeile des Typs der ersten Gruppe ein Dynpro addressiert und dann mit Zeilen des Typs der zweiten Gruppe die

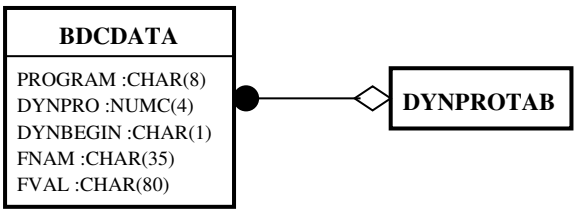


Abbildung 3.15: Eingabedaten der Batch Input-Steuerung für SAP-Transaktionen

Prozessierung der Felder des Dynpros definiert wird. Die Bedeutung der zulässigen Feldwerte würde diese Darstellung sprengen und kann in [SAP96a] nachgelesen werden.

Die erste Methode der indirekten Ausführung der SAP-Transaktionen über die Hintergrundsteuerung benutzt sogenannte *Batch Input-Mappen*. Eine Batch Input-Mappe enthält die notwendige Zugangsinformationen bestehend aus Mandant und Benutzernamen zuzüglich Mappennamens, eines Sperrdatums, das einen frühesten Bearbeitungszeitpunkt angibt, und eines Flags, das die Löschung der Mappe nach der Verarbeitung bestimmt. Eine Funktionsbausteingruppe zum Anlegen oder Öffnen neuer Mappen, Einfügen von SAP-Transaktionen und Schließen geöffneter Mappen ermöglicht die Verwaltung:

`BDC_OPEN_GROUP` Anlegen bzw. Öffnen einer Batch Input-Mappe. Parameter:

- `CLIENT` Mandant der Ausführung
- `GROUP` Mappenname
- `HOLDDATE` Sperrdatum, das den frühest möglichen Ausführungszeitpunkt angibt
- `KEEP` Flag, das bestimmt, ob die Mappe nach der Verarbeitung gelöscht wird
- `USER` Benutzer der Ausführung

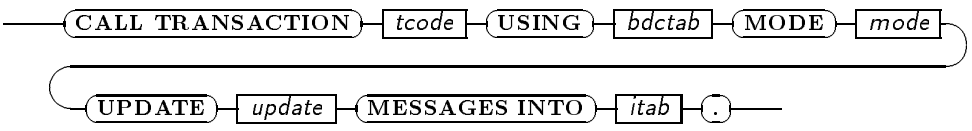
`BDC_INSERT` Einfügen einer Transaktion in die augenblicklich geöffnete Mappe. Parameter:

- `TCODE` Transaktioncode
- `DYNPROTAB` Tabelle der Dynpro-Eingabedaten

`BDC_CLOSE_GROUP` Schließen der aktuell geöffneten Mappe

Durch das ABAP/4-Programm `RSBDCSUB` werden die Batch Input-Mappen ausgeführt. Es überprüft alle registrierten Mappen, ob ihr Sperrdatum bereits abgelaufen ist und beginnt dann für diese Mappen die Verarbeitung der zugehörigen SAP-Transaktionen. Es ist sinnvoll das Progamm `RSBDCSUB` in der Hintergrundsteuerung des SAP-Systems zu registrieren, damit es in regelmäßigen Abständen den Bestand der Mappen prüft und solche mit abgelaufenem Sperrdatum ausführt.

Als zweite Möglichkeit des Batch Input gibt es zwei ABAP/4-Befehle, um SAP-Transaktionen direkt aufzurufen. Sie können entweder innerhalb einer neuen eingeschachtelten SAP-LUW durch



oder in der laufenden SAP-LUW durch den Aufruf

ausgeführt werden. Ähnlich dem Verfahren bei Batch Input-Mappen wird die SAP-Transaktion `tcode` durch die Tabelle der Dynpro-Eingabedaten `bdctab` automatisch gesteuert ausgeführt. Dabei wird die Verarbeitung durch die Parameter bestimmt:

	Mappen	ABAP/4-Aufruf
Verarbeitung	asynchron	synchron
Transaktionen	mehrere	eine
Verbuchung	synchron	synchron/synchron
Debugging	Protokoll	Rückgabewerte/Fehlertabelle
Perfomanz	niedrig	hoch

Tabelle 3.2: Vergleich der Methoden des Batch Input

`MODE` Bestimmt den Anzeigemodus während der automatischen Verarbeitung: `A` – alles Anzeigen, `N` – nichts Anzeigen, `E` – nur Fehler werden Angezeigt. Wenn der Aufruf der SAP-Transaktion aus einem entfernt aufgerufenem Funktionsbaustein stattfindet, sollte der Modus N sein, weil eventuell kein SAPGUI zur Anzeige verfügbar ist.

`UPDATE` Legt die Art der Verbuchung fest: `S` – asynchron, `A` – synchron. Die asynchrone Verbuchung ist problematisch, weil keine Fehler erzeugt werden. Dadurch ist dem Aufrufer nicht bekannt, ob die Datenbankänderungen tatsächlich durchgeführt wurden.

`MESSAGES INTO` erlaubt die Angabe einer internen Tabelle zur Aufnahme von Verarbeitungsfehlern.

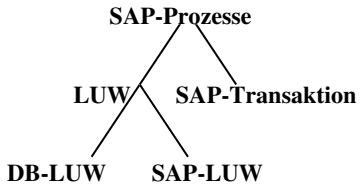
Tabelle 3.2 enthält eine Zusammenfassung der Unterschiede zwischen Batch Input-Mappen und direktem Aufruf.

Damit sind die wesentlichen Konzepte des SAP R/3-Typsystems und der Datenverarbeitung durch ABAP/4-Programme und SAP-Transaktionen erklärt. Der Schluß dieses Kapitels befaßt sich mit dem Transaktionenmodell des SAP R/3 im Sinne atomarer Anweisungen. Dabei wird zum einen ein Bezug zwischen den semantischen Objekten Tabelle, ABAP/4-Programm, SAP-Transaktionen und den Transaktionen verschiedenen Abstraktionsgrades hergestellt. Zum anderen ist aber gerade die transaktionsgesicherte Programmierung ein wichtiger Aspekt der Verteilung in SAP R/3 und somit die Grundlage für die nachfolgende Darstellung der Funktionsbibliothek in Kapitel 4.

3.2.5 Transaktionen in SAP R/3

Die Durchführung von Datenbankmutationen[27] in SAP R/3 birgt diverse Risiken. Alle Mutationen des Datenbestandes müssen in einer solchen Art und Weise durchgeführt werden, daß die Integritätsregeln des Unternehmensdatenmodells nicht verletzt werden. Denn letztendlich ist eine unsichere Veränderung des Datenbestandes eine Verschlechterung gegenüber keiner Veränderung. Entscheidend beim datenbankverändernden Zugriff auf das SAP-System ist die transaktionale Sicherheit im Sinn der Datenbanktheorie (→[GR93, VGH93]). Um diese sicherzustellen muß Klarheit über den Abstraktionsgrad der Datenbankschnittstelle und den damit zusammenhängenden Ereignissen und Befehlen geschaffen werden.

SAP R/3 besitzt ein dreistufiges Transaktionskonzept. Grunsätzlich wird zwischen SAP-Transaktionen und *Logical Units of Work*[28](LUW) unterschieden. Die SAP-Transationen entsprechen einem Modell, um umfangreiche Benutzerdialoge aus vielen Teilschritten abzuwickeln. Im Unterschied dazu dienen LUWs der atomaren und dauerhaften Durchführung von Prozessen. Da der Applikationsserver selbständig LUWs verwalten kann, wird zwischen den LUWs der Datenbank und den SAP-LUWs des Applikationsservers unterschieden.



Die Datenbank-LUW entspricht einer flachen Datenbanktransaktion mit abschließendem Commit. Datenbank-LUWs gewährleisten die Integritätsregeln, die das Datenmodell der Datenbank hergibt.

[27] [Zeh85] folgend wird unter einer Mutation die konsistenzignorierende Veränderung des Datenbestands verstanden.

SAP-R/3 hat ein eigenes Datenmodell mit einem eigenen Typsystem, eigenen Tabellen und einem eigenem Fremdschlüsselbegriff. Die Abbildung dieser SAP-typischen Objekte auf die unterliegende Datenbank, wird von dem Datenbankprozessor vorgenommen, der SAP-Tabellen in Datenbanktabellen und ABAP/4-Typen in die Typen des Datenbanksystems überführt. Viele Integritätsregeln des SAP-R/3, wie z.B. Fremdschlüssel, werden aber garnicht vom Datenbankprozessor verarbeitet. Daher bieten die Datenbank-LUWs nur einen Schutz für skalare Datenwerte. Datenbank-LUWs werden vom SAP-System automatisch bei jedem Bildschirm-Wechsel, also dem Übergang von einem Dynpro auf das nächste, mit einem COMMIT abgeschlossen. Deshalb sind Datenbankänderungen, die mit Native-SQL-Anweisungen durchgeführt werden, spätestens nach einem Dialogschritt persistent.

Die SAP-LUW ist eine flache Transaktion auf dem Niveau des SAP-Applikationsservers. Alle datenbankändernden Open-SQL Anweisungen erzeugen Änderungsaufträge innerhalb der aktuellen SAP-LUW. Diese Änderungsaufträge werden in SAP-Tabellen zwischengespeichert. Die nächste `COMMIT WORK`-Anweisung erzeugt hieraus einen Verbuchungsauftrag, der durch einen Verbuchungs-Workprozeß ausgeführt wird. Ein `ROLLBACK-WORK` würde alternativ die aufgelaufenen Datenbankänderungen verwerfen. Nach einem `COMMIT WORK` wird automatisch auch für die unterliegende Datenbank-LUW ein Commit ausgelöst. Die SAP-LUW ist innhalb des Systems R/3 die eigentlich sichere Aktion bezüglich Atomarität und Dauerhaftigkeit. Isolation zu anderen konkurrent ablaufenden SAP-LUWs muß hingegen durch die explizite Synchronisation über Sperrobjekte realisiert werden. Ähnliches gilt für die Konsistenz, insbesondere weil wichtige Integritätsbedingungen, wie Fremdschlüsselverknüpfungen zwischen Tabellen im Data Dictionary, nicht auf der programmtechnischen Ebene überprüft werden.

Die SAP-Transaktionen wurden als semantisch reichhaltigste Objektklasse der Programmierung bereits in Abschnitt 3.1.2 auf Seite 31 vorgestellt. In dem vorliegenden Zusammenhang repräsentieren sie diejenigen Transaktionen, die der Benutzer auf dem System durchführt. Solche Benutzerdialoge können abhängig von der Anwendung sehr aufwendig und langwierig sein. Die Gewährleistung der Atomarität derartig langer Transaktionen würde im Fall des Transaktionsabbruchs eventuell unnötig viel geleistete Arbeit zunichte machen. Ähnlich würde deren Isolation Sperren über einen zu langen Zeitraum aufrecht erhalten. Deshalb erfüllen die SAP-Transaktionen nicht die Eigenschaften Atomarität und Isolation. Das Modell der SAP-Transaktionen entspricht daher weitgehend dem der „Open Nested Transaction“(→[GR93]). Sie sind ineinanderverschachtelte Transaktionen, über deren strukturelle bzw. dynamische Korrelation keine grundsätzlichen Annahmen gemacht werden. Eine Subtransaktion kann synchron oder asynchron ablaufen. Der erfolgreiche Abschluß einer Subtransaktion kann externe Gültigkeit bekommen, bevor die zugehörige Vatertransaktion abgeschlossen ist. Es wird kein Log geführt, um im Falle des Scheiterns der Vatertransaktion die Subtransaktionen kompensierend rückgängig zu machen. Zusammenfassend verhalten sich SAP-Transaktionen anarchisch, das heißt, daß besonders erwünschte Eigenschaften, wie Atomarität, Isolation oder das abhängige COMMIT einer Subtransaktion vom COMMIT der Vatertransaktion, durch den Programmierer realisiert werden müssen.

Pragmatischen Aspekte der Transaktionsprogrammierung

Die Realisierung von Transaktionen, Mechanismen und damit zusammenhängende Probleme sollen im folgenden vorgestellt werden. Die SAP-LUW kapselt datenverändernde Anweisungssequenzen, bestehend aus Open-SQL Befehlen. Was aber ist der genaue Zusammenhang von SAP-Transaktionen und SAP-LUWs und wie funktioniert die Programmierung mit SAP-LUWs. Außerdem wird die Bedeutung der SAP-LUW für die verteilte Client/Server-Programmierung durch den Remote Function Call analysiert.

Eine SAP-Transaktion startet immer mit einer eigenen SAP-LUW. Deshalb erzeugt der Aufruf `CALL TRANSACTION <tcode>` eine neue SAP-LUW. Die kann dann mit den Open-SQL Befehlen `COMMIT WORK` oder `ROLLBACK WORK` in einem ihrer Module auf der Ebene von ABAP/4-Code entweder abgeschlossen oder abgebrochen werden. Dadurch befinden sich allerdings die darauf folgenden Open-SQL

Anweisungen in einer neuen SAP-LUW innerhalb der alten SAP-Transaktion. Dies entspricht fast dem Modell der verketteten Transaktionen, abgesehen davon, daß eine Anweisung `CHAIN WORK` nicht vorhanden ist und somit bei jedem Übergang in eine neue SAP-LUW der Datenbankkontext inklusive aller Cursor verloren geht. Weiterhin können SAP-Transaktionen mit dem Aufruf `CALL TRANSACTION <tcode>` geschachtelt werden. Mit `LEAVE` wird die aktuelle SAP-Transaktion abgebrochen und durch `LEAVE TO TRANSACTION <tcode>` wird gleichzeitig eine neue SAP-Transaktion gestartet.

Die Ausführung von ABAP/4-Programmen und SAP-Transaktionen ist strikt sequentiell. Das bedeutet, daß eine ABAP/4-Anweisung erst dann interpretiert wird, nachdem die Ausführung ihres Vorgängers beendet ist. Ebenso wird ein Nachfolge-Dynpro einer SAP-Transaktion erst dann aktiviert, wenn die Postprozessierung des Vorgänger-Dynpros abgeschlossen ist. Um Nebenläufigkeit in diese strikt serielle Ausführungsreihenfolge hineinzubekommen, können asynchron aufgerufene Funktionsbausteine (vgl. Abschnitt 4.1.2) oder der Batch Input-Mechanismus (vgl. Seite 3.2.4) verwendet werden. Letzterer erzeugt einen neuen Auftrag für die Bearbeitung einer SAP-Transaktion, der dann asynchron durch die Hintergrundverarbeitung ausgeführt wird. Der präferierbare Mechanismus sind die asynchronen Funktionsbausteine, die in Kombination mit dem Aufruf `CALL TRANSACTION` auch zum nebenläufigen Ausführen von SAP-Transaktionen geeignet sind.

Es wird deutlich, daß zwar semantisch die Blöcke der Abläufe genau durch die SAP-Transaktionen dargestellt werden, aber die Basiseinheiten unteilbarer Anweisungen SAP-LUWs sind. Die Programmierung für SAP-LUWs wird durch die Anweisung `PERFORM <form> ON COMMIT` speziell unterstützt. Sie erlaubt es, Programmteile in ein Unterprogramm `<form>` auszulagern, das dann erst beim nächsten `COMMIT WORK` ausgeführt wird. Transaktionale Programmanteile mit irreversiblen Auswirkungen können auf diese Weise von der tatsächlichen Durchführung der SAP-LUW abhängig gemacht werden. Denn wenn ein Abbruch der SAP-LUW stattfindet wird der Aufruf einfach vom System gelöscht und damit niemals ausgeführt. In Abschnitt 4.1.2 wird gezeigt, daß der gleiche Mechanismus bei dem transaktionsgesicherten Remote Function Call eine Ausführung verhindert, wenn die SAP-LUW des Aufruferkontexts nicht mit einem `COMMIT WORK` abgeschlossen wird.

3.3 Zusammenfassung

Die Systemarchitektur des SAP R/3 orientiert sich an „3-tier“[29]-Client/Server-Systemen mit ihrer Trennung zwischen Präsentation-, Applikations- und Datenbankdiensten. Die komplexen Zusammenhänge der Prozeßarchitektur der Applikationsdienste wurden in einem Objektmodell evaluiert.

Die semantischen Objekte des SAP R/3 sind SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen. Anhand der dreischichtigen Architektur können Schnittstellen auf ihre Offenheit gegenüber diesen semantischen Objekte untersucht werden. Der Remote Function Call zum Aufruf von Funktionsbausteinen ermöglicht den Zugriff auf SAP-Tabellen, die dynamische Modifikation und Ausführung von ABAP/4-Programmen und den Aufruf von SAP-Transaktionen. Funktionsbausteine sind insbesondere selbst spezielle ABAP/4-Programme.

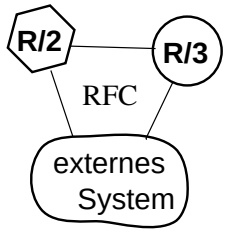
Grundlage der Programmierung mit Funktionsbausteinen sind neben der Sprache ABAP/4 die Prozesse der SAP-Anwendungen. Sie bestehen aus „anarchischen“ SAP-Transaktionen, atomaren und Dauerhaftigkeit gewährleistenden SAP-LUWs auf Applikationsserverebene und den Datenbank-Transaktionen. Im folgenden Kapitel 4 wird der Zusammenhang zwischen SAP-LUWs und Funktionsbausteinen weiter vertieft.

[29] vergleiche Seite 7

Kapitel 4

Die SAP R/3-Funktionsbibliothek

SAP R/3 ist ein verteiltes Client/Server-System für betriebswirtschaftliche Anwendungen. Spezielle ABAP/4-Programme, die *Funktionsbausteine*, bilden die Schlüsseltechnologie der Verteilung. Funktionsbausteine sind Kommunikationsendpunkte(Service-Funktionen) für Funktionsaufrufe der Sprache ABAP/4 oder externer Systeme. Funktionsbausteine werden durch aus ABAP/4-Programmen oder von externen Systemen durch den Remote Function Call ausgerufen.



Dadurch, daß SAP RFC-Programmierschnittstellen für die Systeme R/2, R/3 und externe Systeme anbietet und der RFC eine „highlevel"-Schnittstelle bzgl. der Möglichkeit, durch ihn mit einer Funktionsaufrufsemantik entfernt zu kommunizieren, ist, kann er als die wichtigste Kommunikationsschnittstelle des SAP R/3-Systems angesehen werden. Dies wird unter anderem dadurch belegt, daß viele weitere Kommunikationsprotokolle des R/3, wie z.B. das ALE (siehe Seite 37), auf den RFC als Basisprotokoll aufsetzen.

Der Remote Function Call ist die Einbettung eine Remote Procedure Call(RPC)-Dienstes in die Sprache ABAP/4. Die wesentlichen Komponenten der RPC-Kommunikation werden in [BEG96] genannt:

Stubs Schnittstellenmodule zwischen Client bzw. Server und dem Laufzeitsystem.

Binding Lokalisierung eines Servers durch den Client.

Data Representation Gemeinsame Darstellung der Daten für Client und Server.

Transport Protocol Transportdienst über das Netz.

Control Protocol Statusverfolgung für RPC-Aufrufe.

Von diesen Komponenten sind beim Remote Function Call nur die letzteren vier Relevant. Stubs werden deshalb nicht gebraucht, weil alle Aufrufe über eine interne dynamische Schnittstelle des Systems abgewickelt werden. Die RFC-Bindung bezieht ihre Adresse aus dem Namen des aufgerufenen Funktionsbausteins und einer logischen Adresse, die einen Eintrag der SAP-Tabelle `RFCDES` identifiziert. Die Datenrepräsentation wird von der Laufzeitumgebung angepaßt. Die Statusverfolgung ist einerseits durch die Generierung von Ausnahmen auf der Basis des zugrundeliegenden Request/Reply-Protokolls (→[CDK94]) sichergestellt und andererseits gibt es für Aufrufe, die der Verbuchung dienen, die Möglichkeit den Erfolg eines Aufrufs über das Monitoring-System des SAP R/3 zu überprüfen.

Nach [CDK94] gibt es zwei Klassen von RPC-Protokollen:

- Die erste Klasse beinhaltet solche Systeme deren RPC-Mechanismus in das System integriert ist und deren Systemsprache eine spezielle Notation zur Definition von entfernten Prozeduraufrufen enthält.

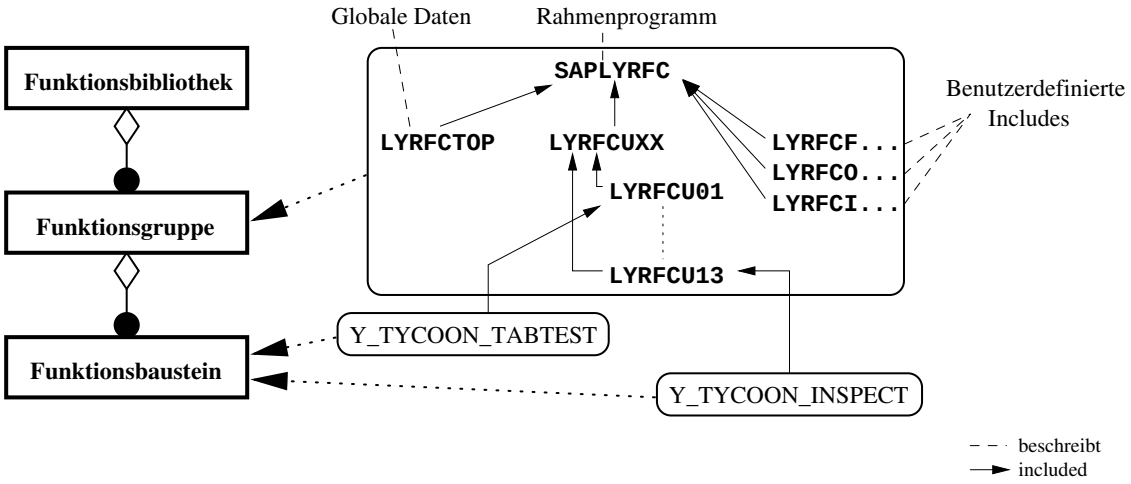


Abbildung 4.1: Struktur der Funktionsbiblithek

- Die zweite Klasse bedient sich einer speziellen Sprache zur Schnittstellenbeschreibung zwischen Client und Server.

Die Sprache zur Schnittstellenbeschreibung des SAP-RFC ist Client-seitig ein spezielles ABAP/4-Kommando(siehe Abschnitt 4.1.2). Die Beschreibung der Funktionsbausteinschnittstellen, als Kommunikationsendpunkte und Service-Funktionen des RFC-Aufrufs, wird über spezielle SAP-Transaktionen (siehe Abschnitt 4.1.1) vorgenommen. Damit fällt der Remote Function Call in die Klasse der integrierten RPCs.

Dieses Kapitel befaßt sich mit der *Funktionsbibliothek* als Teil eines SAP R/3-Applikationsservers und der SAP R/3-Workbench. Die Funktionsbibliothek beinhaltet die Menge aller Funktionsbausteine eines SAP-Applikationsservers. Wichtige Aspekte der Funktionsbibliothek sind die Verwaltung der Funktionsbausteine, die Semantik der Funktionsbausteinaufrufe, insbesondere die transaktionale Absicherung, und die Funktionsbausteinschnittstelle für externe Systeme. Es wird im folgenden zunächst die Organisation der Funktionsbibliothek und die RFC-Programmierung in ABAP/4 erläutert. In einem zweiten Abschnitt folgt die Analyse der RFC-Bibliothek für externe Systeme (vgl. Abschnitt 4.2).

4.1 Organisation und Verwaltung der Funktionsbausteine

Die Funktionsbibliothek als Aggregation aller Funktionsbausteine eines Applikationsservers partitioniert die die Funktionsbausteine entsprechend ihrer Aufgabe in disjunkte Funktionsgruppen (→[CPG96]). Die Struktur entspricht der in Abbildung 4.1 wiedergegebenen dreistufigen Hierarchie. SAP-intern sind die Funktionsbausteine in Include-Programme aufgeteilt. Ein zugehöriges Rahmenprogramm nimmt ein Toplevel-Include zuzüglich eines weiteren Includes auf, das seinerseits alle Funktionsbaustein-Includes einbezieht. Dabei werden die Bezeichner für die Include-Programme wie in Abbildung 4.1 gewählt. Das Rahmenprogramm ist die Übersetzungseinheit für alle seine Funktionsbausteine und repräsentiert gleichzeitig die Funktionsgruppe. Die Übersetzungseinheit wird durch die Schlüsselworte `FUNCTION POOL <group>` im Toplevel-Include eingeleitet. Jeder Funktionsbaustein beginnt mit `FUNCTION` und endet mit `ENDFUNCTION`.

Verglichen mit anderen ABAP/4-Programmen, wie Reports und Modulpools und den dort enthaltenen ABAP/4-Unterprogrammen (vgl. Abschnitt 3.2.3), liegt ein Vorteil der Funktionsbausteine in ihrer zentralen Verwaltung innerhalb der Funktionsbibliothek des Applikationsservers und ihrer explizit verwalteten Schnittstelle. Diese Schnittstelle abstrahiert von der Implementation und ermöglicht den entfernten Aufruf über Systemgrenzen hinweg.

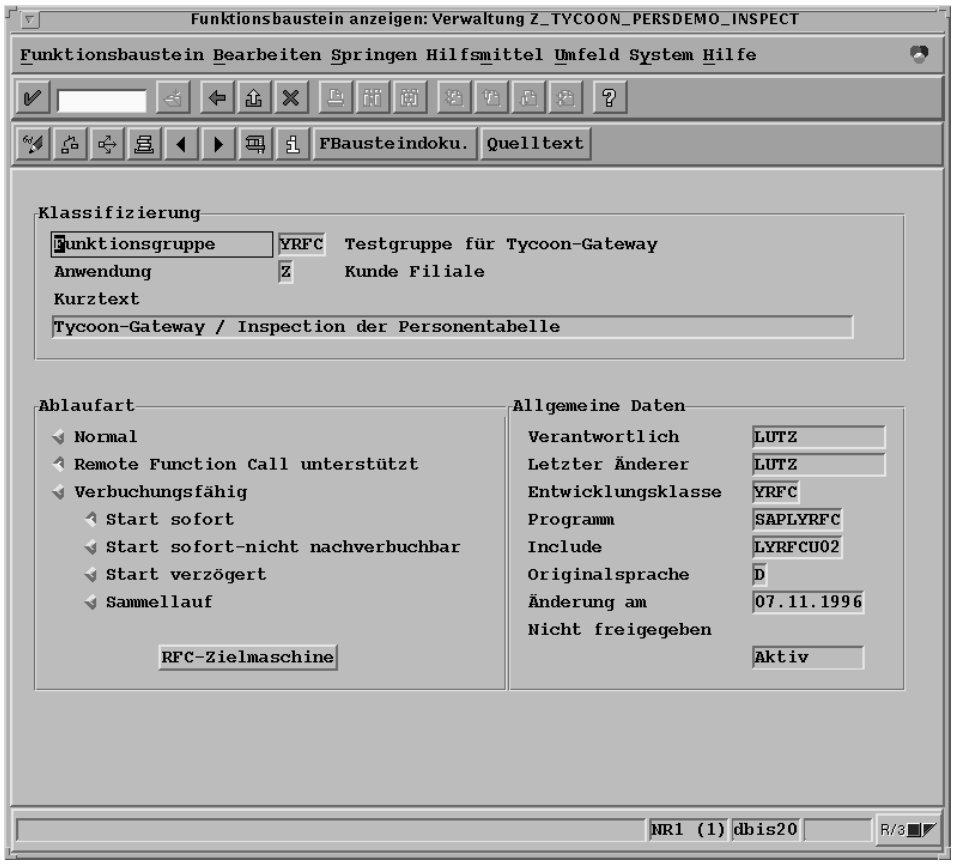


Abbildung 4.2: Verwaltung für Funktionsbausteine

Die Verwaltung der Funktionsbausteine wird über die Workbench durch das in Abbildung 4.2 dargestellte Dynpro vorgenommen. Es enthält eine Klassifizierung, Ablaufart und allgemeine Daten. Zusätzlich können Funktionsbausteine individuell dokumentiert und versioniert werden.

Die Klassifizierung besteht aus der Funktionsgruppe, einer Anwendungart und einem beschreibenden Kurztext. Die Anwendungsart bestimmt das Arbeitsgebiet und wird in der Regel freigelassen oder mit '*' als neutral gekennzeichnet (→[SAP96b]). Die Ablaufart bestimmt die Datenbankfortschreibungsart oder aktiviert die Möglichkeit des entfernten Zugriff auf einen Funktionsbaustein. Hier stehen folgende Möglichkeiten zur Auswahl (→[Mat96b]):

Normal Ausschließlich lokale Aufrufe sind möglich. Der Funktionsbaustein verhält sich, wie ein normales Unterprogramm.

Remote Function Call unterstützt Entfernte Funktionsbausteinaufrufe werden unterstützt.

Verbuchungsfähig Der Funktionsbaustein ist ein Verbuchungsbaustein, der mit `IN UPDATE TASK` aufgerufen wird (siehe Seite 67). Die Art der Verbuchung wird durch einen der folgenden Schalter bestimmt:

- **Start sofort** V1-Verbuchung; die Verbuchung beginnt sofort. Wenn ein Verbuchungsvorgang nicht durchgeführt werden kann, weil eventuell Systemsperren dies verhindern, dann wird dieser Versuch automatisch wiederholt.
- **Start sofort – nicht nachverbuchbar** V1-Verbuchung; die Verbuchung beginnt sofort. Fehlgeschlagene Verbuchungsversuche werden nicht wiederholt.
- **Start verzögert** V2-Verbuchung; die Verbuchung ist nicht zeitkritisch und kann deshalb verzögert beginnen. Alle V2-Verbuchungen laufen unabhängig voneinander ab.
- **Sammellauf** Die Verbuchung ist nicht zeitkritisch und wird aufgehoben, bis der Benutzer einen expliziten Verbuchungslauf startet.

Allgemeine Daten eines Funktionsbausteins sind Angaben über den Verwalter, den letzten Änderer, die Entwicklungsklasse, das Rahmenprogramm, das zugehörige Include-Programm, die Sprache, das Datum des letzten Änderns und einen Freigabeflag. Das Freigabeflag bestimmt, ob andere Entwickler als der Verwalter des Funktionsbausteins Änderungen vornehmen dürfen.

4.1.1 Schnittstellendefinition

Die Aufrufschnittstelle eines Funktionsbausteins setzt sich aus Eingabeparametern, Ausgabeparametern, Changing-Parametern, Tabellenparametern und Ausnahmen zusammen. Die Spezifikation der Parameter ist über zwei Dynpros verteilt.

In Abbildung 4.3 ist das Dynpro für Eingabeparameter, Ausgabeparameter und Changing-Parameter dargestellt. Die Eingabeparameter werden unter `IMPORTING`, Ausgabeparameter unter `EXPORTING` aufgeführt[1] `CHANGING`-Parameter sind sowohl Aufruf- als auch Ergebnisparameter. Die voreingestellte Parametersemantik ist „call by value“. Das gilt auch für `CHANGING`-Parameter, die lediglich bei Terminieren der Funktion eine Rückübertragung der Werte veranlassen. Alle drei Parameterarten können als Referenzparameter, also „call by reference“, gekennzeichnet werden. Dadurch würden Import- und Changing-Parameter in ihrer Semantik übereinstimmen. Der Unterschied zwischen Changing- und normalen Referenz-Parametern liegt im Abbruchfall. Während bei Referenz-Parametern Wertänderungen auf jeden Fall auch außerhalb der Funktion sichtbar werden, findet für Changing-Parameter die Wertrückübertragung nur dann statt, wenn die Funktion ohne Abbruch durch das Auslösen einer Ausnahme beendet wird. Zur Angabe der Paramtertypen können entweder Strukturen oder Felder von SAP-Tabellen angegeben werden oder es wird ein Typ eines Typen-Pools (vlg. Abschnitt 3.2.2) der lokalen Funktionsgruppe verwendet.

[1] SAP argumentiert, daß die Bezeichnungen `IMPORTING` bzw. `EXPORTING` in der Schnittstelle die Assoziation des Importierens der Aufrufparameter bzw. des Exportierens der Ergebnisparameter herstellen.

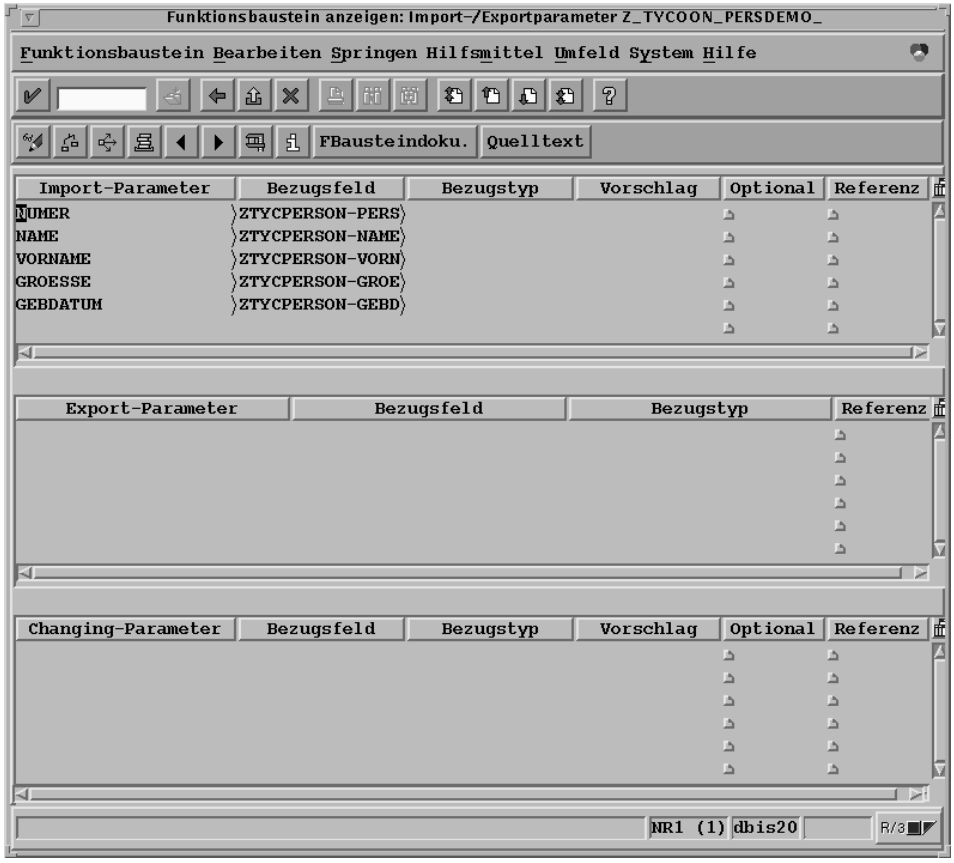


Abbildung 4.3: Import-, Export- und Changing-Parameter von Funktionsbausteinen

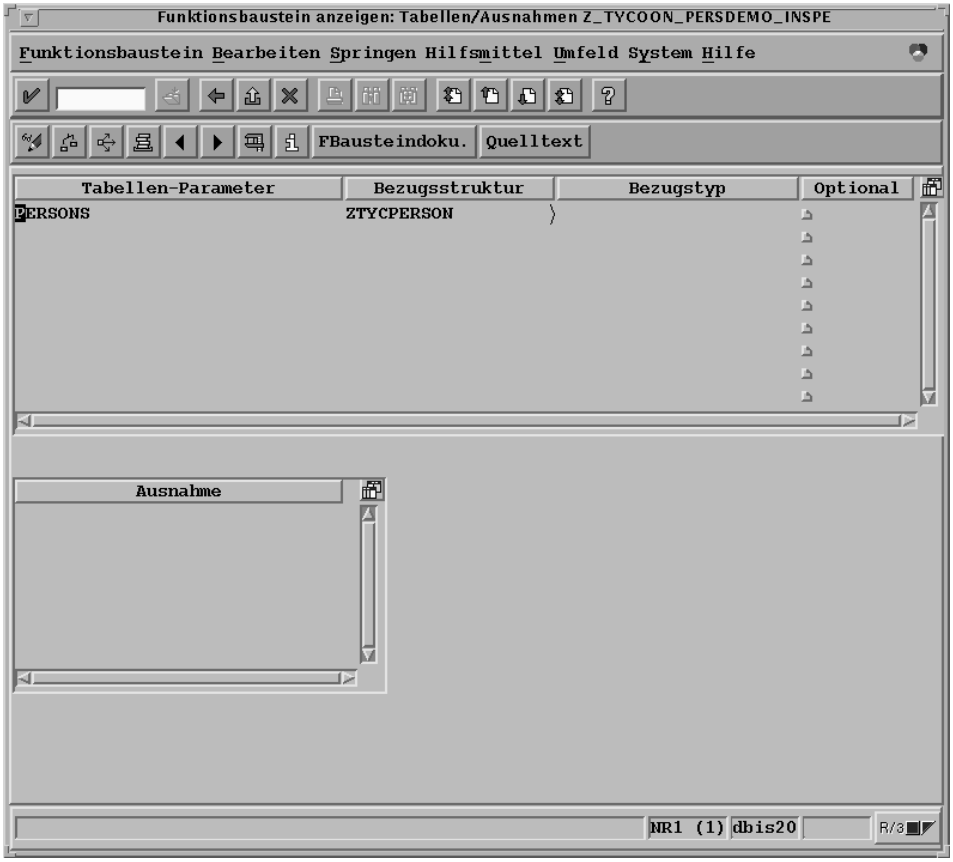
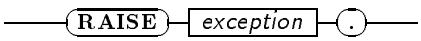


Abbildung 4.4: Tabellen-Parameter und Ausnahmen von Funktionsbausteinen

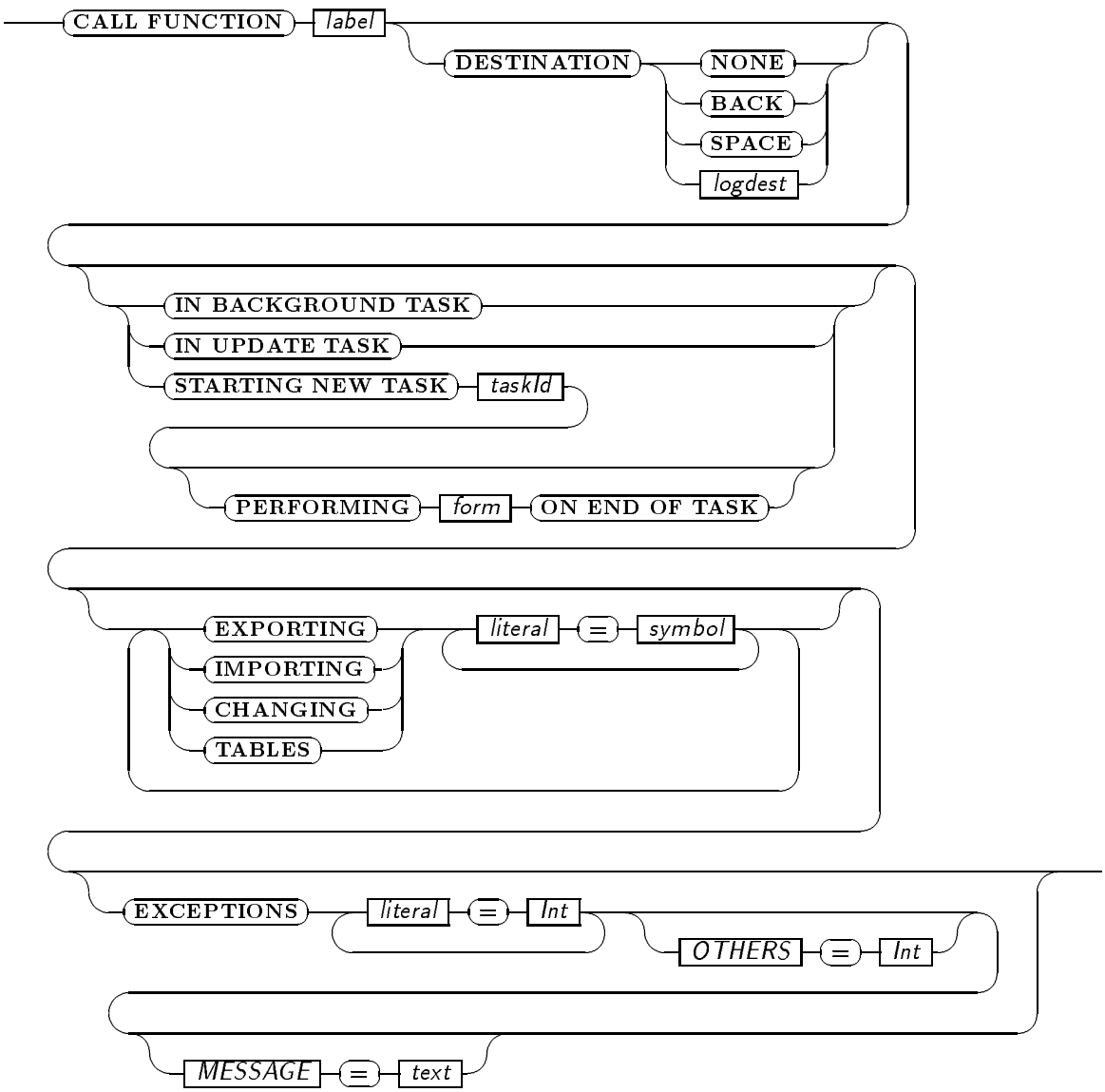
Abbildung 4.4 enthält das Dynpro für die Tabellenparameter und Ausnahmen. Wie bei den übrigen Parametern sind hier als Typen sowohl einzelne Bezugsfelder, Strukturen oder ABAP/4-Typdeklarationen eines Typen-Pools der lokalen Funktionsgruppe zugelassen. Ausnahmen werden durch Zeichenketten benannt. Sie können darauhin im Quelltext der Funktion durch



ausgelöst werden, müssen aber beim Aufrufer durch Zuweisung zu einer ganzen Zahl über dieselbe als Rückgabewert identifiziert werden.

4.1.2 Aufruf- und Parameterübergabekonventionen

Syntaktisch ist der RFC mit folgender ABAP/4 Anweisung abgehandelt (→[RFC96c]):



Der minimale Aufruf besteht aus `CALL FUNCTION <label>`. Er ruft einen Funktionsbaustein ohne Ausnahmen und Parameter im lokalen Applikationsserver auf.

Durch den Zusatz `DESTINATION <dest>` kann ein entfernter Applikationsserver angesprochen werden. Dabei gibt es vier Möglichkeiten:

`'NONE'` Der Aufruf wirkt lokal, wird aber wie ein entfernter Aufruf behandelt. Dies steht im direkten Zusammenhang mit der Transaktionssteuerung, da bei einem entfernten Aufruf im Zielsystem eine neue eingeschachtelte SAP-LUW erzeugt wird.

`'BACK'` Dieser Parameter ist nur innerhalb von Funktionsbausteinen zulässig. Er addressiert den Aufrufer des Funktionsbausteins, wodurch dieser zurückgerufen wird.

`'SPACE'` Der Aufruf mit `'SPACE'` wirkt, wie ein Aufruf ohne `DESTINATION`.

`logdest` Hier kann ein logischer Zielname für den Funktionsaufruf angegeben werden. Dieser muß aber in der Tabelle `RFCDES` mit einem konkreten Ziel vermerkt sein.

Die nächste Gruppe Schlüsselwörter, bestehend aus `IN BACKGROUND TASK`, `IN UPDATE TASK` und `STARTING NEW TASK`, bestimmt die Aufrufsemantik und unterstützt das Verbuchungskonzept. Insbesondere müssen beim Zugriff auf Funktionsbausteine unerwünschte Seiteneffekte ausgeschlossen werden. Deshalb gibt es für nicht idempotente[2] Funktionen, wie z.B. das Inkrementieren eines Zählers, Aufrufmöglichkeiten mit „exactly once"-Semantik durch `IN BACKGROUND TASK` und `IN UPDATE TASK`. Dagegen reicht für idempotente Funktionen oder solche mit reinem Berichtscharakter, die keine Veränderungen an der Datenbank durchführen, ein Aufruf mit der Semantik „at least once" aus.

Durch `EXPORTING`, `IMPORTING`, `CHANGING` und `TABLES` werden die Aktualparameter übergeben. Die hier angegebenen Parameter müssen mit Ausnahme der als optional gekennzeichneten Parameter der Schnittstellenspezifikation(vgl. Abschnitt 4.1.1) des aufgerufenen Funktionsbausteins entsprechen. Bezüglich des Syntaxdiagramms gibt es hier einge Einschränkungen, da abhängig vom Aufruf nicht immer alle Parameterarten genutzt werden können. Zunächst gibt ein asynchron zur Verbuchung aufgerufener Funktionsbaustein niemals Ergebnisparameter zurück. Desweiteren erfolgt die Zuordnung der Ein/Ausgabeparameter beim Funktionsbausteinaufruf genau umgekehrt gegenüber der Deklaration in der Schnittstelle[3], so daß `IMPORTING`-Parameter des Aufrufs auf `EXPORTING`-Parameter der Schnittstelle abgebildet werden und umgekehrt.

Ausnahmen können durch Zuweisungen hinter dem Schlüsselwort `EXCEPTIONS` ganzzahlige Werte zugeordnet werden. Die Systemvariable `SY-SUBRC` enthält dann bei Auftreten einer Ausnahme den entsprechenden Wert und kann nach dem Aufruf direkt geprüft werden. Dabei gibt es zwei vordefinierte Ausnahmen:

`SYSTEM_FAILURE` Hiermit werden Fehler im entfernten System angezeigt.

`COMMUNICATION_FAILURE` Diese Ausnahme wird durch Fehler der Kommunikation erzeugt.

Durch Nachstellen von `MESSAGE <literal>` kann die jeweils bei einem Auftreten solch einer Ausnahme erzeugte Textmeldung eingelesen werden.

Wegen ihrer Besonderheiten und über eine kurze Erwähnung hinausgehenden Eigenschaften wird im folgenden auf Tabellenparameter, Asynchrone RFCs und Transaktionale RFCs, sowie die Verbuchung gesondert eingegangen.

Tabellenparameter

Tabellenparameter erlauben die Übergabe interner Tabellen (vgl. Abschnitt 3.2.3) als Parameter eines RFC. Allerdings unterliegen diese einer speziellen Semantik, die von der Art des Aufrufs abhängig ist.

[2] Funktionen mit der Eigenschaft: $f(x) = f(f(x))$

[3] SAP argumentiert, daß die Bezeichnungen `EXPORTING` bzw. `IMPORTING` bei einem Aufruf die Assoziation des Exportierens der Aufrufparameter aus dem Aufruferkontext bzw. des Importierens der Ergebnisparameter in den Aufruferkontext herstellen.

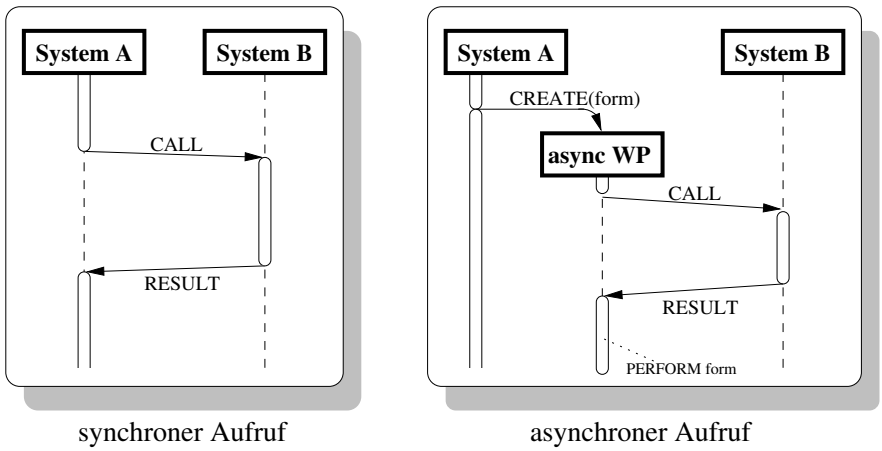


Abbildung 4.5: Vergleich des synchronen mit dem asynchronen RFC

Bei einem lokalen Aufruf, der durch das Aufrufziel `'SPACE'` oder das Fehlen des Schlüsselworts `DESTINATION` durchgeführt wird, wird eine Referenzsemantik benutzt. Das bedeutet, daß der aufgerufene Funktionsbaustein auf dieselbe Tabelle zugreift, wie der Aufrufer. Die Tabellendaten beider Programmkontexte sind deshalb jederzeit vollständig abgeglichen.

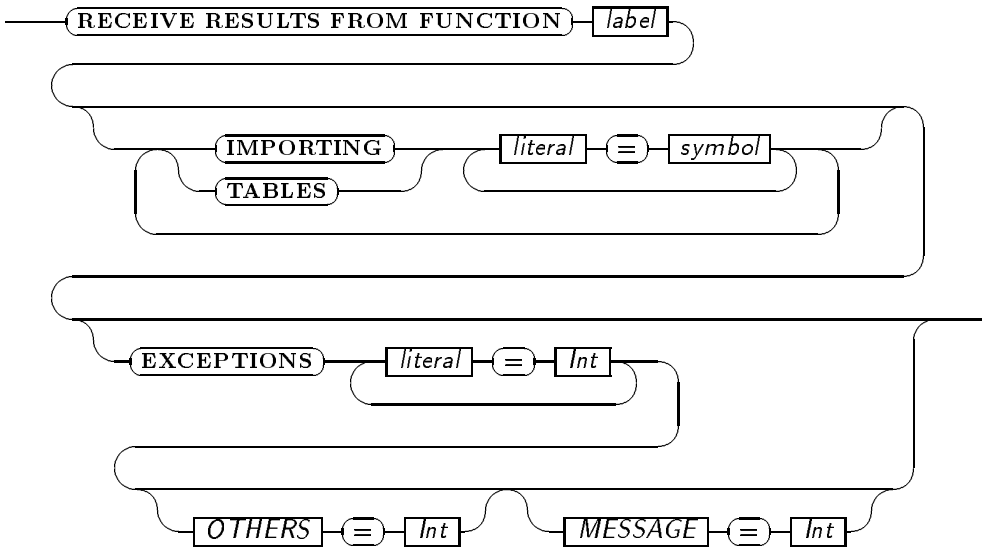
Andernfalls verweist `<logdest>` auf einen Eintrag der Tabelle `RFCDES`(`TRFCD` in R/2) oder enthält den Wert `'NONE'`, so daß ein vermeintlich entferntes System das Ziel des Aufrufes ist. In jedem Fall kommt dann dann der *Delta Manager*(→[WHSH95]) zur Anwendung. Er stellt beim RFC-Aufruf eine Kopie der internen Tabelle des Aufrufers im Rollbereich des aufgerufenen Funktionsbausteins her. Beim Aufrufer wird eine Objekt-ID zur der dann als „virtual global table“ gekennzeichneten Tabelle erzeugt. Alle Änderungen an der Tabelle in der aufgerufenen Funktion werden daraufhin als Log-Information hinterlegt und jeweils beim nächsten Wechsel des Kontrollflusses zum Aufrufer übermittelt. Dies kann entweder durch das Beenden des RFCs, durch eine Ausnahme oder durch einen Rückruf stattfinden, was auch bei mehrfachem Wechsel des Kontrollflusses sicherstellt, daß für Tabellenparameter nur die Änderungen Übertragen werden. Die Objekt-ID wird gelöscht, wenn der RFC beendet wird.Durch diesen Mechanismus, der den Abgleich der Änderungen an der internen Tabelle mit dem Kontrollfluß zwischen Aufrufer und Funktionsbaustein synchronisiert, werden unnötige Netzwerkbelastung durch häufiges Hin- und Herkopieren von Tabellen vermieden.

Tabellenparamter sind grundsätzlich optional. Läßt man sie beim Aufruf eines RFC weg, dann wird im aufgerufenen Funktionsbaustein eine leere Tabelle benutzt.

Asynchrone RFCs

Asynchrone RFCs sind eine Erweiterung des SAP R/3 ab Version 3.0. Im Unterschied zum normalen synchronen Aufruf, siehe Abbildung 4.5, wird der Kontrollfluß des Aufrufers nicht unterbrochen, sondern ein neuer Workprozeß eingerichtet, der den Aufruf asynchron zum Aufrufer durchführt. Das aufrufende Programm läuft weiter. Wenn der neu eingerichtete Workprozeß den Aufruf durchgeführt hat und die Ergebnisparameter vorliegen, ruft er ein ihm übergebenes ABAP/4-Unterprogramm auf, daß die Ergebnisparameter des Aufrufs als Argumente erhält und über globale Variablen in den Kontext des Aufrufers schreiben kann

Der Asynchrone Aufruf wird durch die Schlüsselworte `STARTING NEW TASK <taskId>` eingeleitet. Der Parameter `<taskId>` ist ein String der diesen Aufruf und gleichzeitig damit die Verbindung identifiziert. Das lokale ABAP/4-Unterprogramm `<form>` zur Entgegennahme der Ergebnisparameter kann durch die Schlüsselworte `PERFORMING <form> ON END OF TASK` spezifiziert werden. Diese Unterroutine übernimmt zumindest die `<taskId>` in einem `USING`-Parameter, so daß in dem Unterprogramm eine Zuordnung der Ergebnisparameter zum Aufruf getroffen werden kann. Durch die Anweisung



werden die Ergebnisparameter ermittelt.

Beispiel:

```
CALL FUNCTION 'Z_GET_SECRET_DATA' STARTING NEW TASK 'PERSONAL'
   DESTINATION 'PENTAGON'
   PERFORMING 'RETURN_PERSONAL' ON END OF TASK.


FORM RETURN_PERSONAL USING TASKNAME.

   RECEIVE RESULTS FROM FUNCTION 'Z_GET_SECRET_DATA'
      IMPORTING     FLIGHTID = SFLIGHT-ID
      EXCEPTIONS    SYSTEM_FAILURE MESSAGE SYSTEM_MSG.

ENDFORM.
```

Die Unterroutine `RETURN_PERSONAL` wird ausgeführt, sobald das Ergebnis des Asynchronen RFC vorliegt. Mit der Anweisung `RECEIVE RESULTS FROM` wird der Ergebnisparameter `SFLIGHT-ID` direkt in die Variable `FLIGHTID` gespeichert.

Transaktionale RFCs

Transaktionale RFCs sind, wie der Asynchrone RFC eine Erweiterung des SAP R/3 ab Version 3.0. Transaktionale RFCs haben ebenfalls eine asynchrone Aufrufsemantik, garantieren jedoch im Unterschied zum asynchronen RFC die „exactly once"-Aufrufsemantik. Ein Transaktionaler RFC wird durch die Schlüsselworte `IN BACKGROUND TASK` abgesetzt. Aus den gleichen Gründen wie beim Asynchronen RFC kann der Transaktionale RFC keine Ergebnisparameter entgegennehmen. Hinzu kommt, daß der Aufruf selbst erst beim nächsten `COMMIT WORK`, also mit dem erfolgreichen Abschluß der SAP-LUW des Aufruferkontexts durchgeführt wird. Deshalb sind Rückrufe des aufgerufenen Funktionsbausteins nicht erlaubt.

Weitere wichtige Eigenschaften für Transaktionale RFCs, die innerhalb derselben SAP-LUW aufgerufen werden, sind:

- Die Reihenfolge der Aufrufe beibehalten.

- Im Zielsystem wird ein einziger Workprozess respektive SAP-LUW benutzt und dadurch der Programmkontext, wie globale Variablen, beibehalten.

- Durch die Verwendung einer einzigen SAP-LUW im Zielsystem sind alle Aufrufe in atomarer Weise von einem Abbruch betroffen.

Transaktionale Aufrufe werden in den Tabellen `ARFCSSTATE` und `ARFCSDATA` gespeichert, um beim nächsten Abschluß der laufenden SAP-LUW mit `COMMIT WORK` ausgelesen und ausgeführt zu werden. Ist dann das Zielsystem nicht innerhalb einer bestimmten Zeit erreichbar, wird ein Batch-Auftrag des Reports `RSARFCSE` und einem eindeutigen Transaktionsidentifikator[4] erzeugt. Dieser Report wird einige Perioden wiederholt ausgeführt. Wenn das Zielsystem unerreichbar bleibt, wird nach einer festen Zahl von Perioden die Verarbeitung abgebrochen und der Status `CPICERR` in die Tabelle `ARFCSDATA` eingetragen. Nach einem weiteren Zeitintervall[5] wird der Eintrag aus der Tabelle `ARFCSSTATE` gelöscht.

Die Eigenschaften des Transaktionalen RFCs eignen ihn am besten zur asynchronen verteilten Verbuchung von Daten. Häufig ziehen Dialogtransaktionen umfangreiche Verbuchungen nach sich. Um nun Wartezeiten zwischen den Dynpros zu vermeiden, können solche Verbuchungen in einem Transaktionalen RFC durchgeführt werden. Durch seine „exactly once“-Aufrufsemantik garantiert er, daß die Änderung der Datenbestände genau einmal durchgeführt werden.

Aufruf von Verbuchungsbausteinen

Der Aufruf von Verbuchungsbausteinen stellt das lokale Pendant zum Transaktionalen RFC der verteilten Programmierung dar. Insbesondere ist seine Aufrufsemantik also „exactly once“. Allerdings unterscheidet sich die zeitliche Synchronisation des Aufrufs von der bei asynchronen transaktionalen RFCs. Ein Aufruf kann sowohl als V1-Verbuchung als auch V2-Verbuchung durchgeführt werden. Alle V1-Verbuchungen, die innerhalb derselben SAP-LUW aufgerufen werden, werden sofort bei Beendigung der laufenden SAP-LUW mit `COMMIT WORK` gemeinsam ausgeführt. Ein Abbruch einer dieser V1-Verbuchungen führt zum automatischen Abbruch und Rücksetzen aller V1-Verbuchungen derselben SAP-LUW. Im Unterschied dazu werden V2-Verbuchungen zeitversetzt und mit niedrigerer Priorität ausgeführt. Außerdem hat der Abbruch einer der V2-Verbuchungen weder Einfluß auf die übrigen V2-Verbuchungen noch die V1-Verbuchungen derselben SAP-LUW.

Funktionsbausteinen der Verbuchung muß in ihrer Verwaltungsinformation eine spezielle Ablaufart eingestellt werden (siehe Seite 4.1), die bereits darüber entscheidet, ob es sich um eine V1- oder V2-Verbuchung handelt. Der Aufruf solcher Bausteine wird durch die Schlüsselworte `IN UPDATE TASK` durchgeführt. Weil Verbuchungs-RFCs erst mit dem `COMMIT WORK` der kapselnden SAP-LUW aktiviert werden, sind, wie beim Transaktionalen RFC, keine Ausgabeparameter, Ausnahmen oder Rückrufe möglich.

Zusammenfassung der Aufrufformen und Parameterarten

In Tabelle 4.1 sind noch einmal die Eigenschaften aller Aufrufmöglichkeiten für Funktionsbausteine in ABAP/4 zusammengefaßt. Die Spalte *Entfernt* bedeutet, daß die Angabe eines entfernten Aufrufziels durch `DESTINATION` unterstützt wird. Die Spalte *Gleiche LUW* bedeutet, daß mehrere Aufrufe zum gleichen Zielsystem, die innerhalb einer SAP-LUW abgesetzt werden, innerhalb derselben SAP-LUW im Zielsystem abgearbeitet werden und somit eine atomare Handlung darstellen. Die letzte Zeile nimmt eine Sonderrolle ein, weil die Schlüsselworte `RECEIVE RESULTS FROM` ausschließlich der Entgegennahme von Ergebnisparametern des Asynchronen RFC dienen.

Einen weiteren Überblick über den Zusammenhang zwischen diversen RFC-Aufrufmöglichkeiten, einigen ABAP/4-Anweisungen und dem Transaktionensystem aus Datenbank-LUW, SAP-LUW und SAP-Transaktionen gibt Tabelle 4.2.

[4] Laut SAP sind dies Transaction-IDs weltweit eindeutig.
[5] Voreingestellt sind 8 Tage.

Schlüsselwort	Semantik	Entfernt	Gleiche LUW	IMPORT	EXPORT	CHANGING	TABLE	EXCEPTION	CALLBACK
CALL FUNCTION	exactly once	nein	nein	ja	ja	ja	ja	ja	ja
DESTINATION	at least once	ja	nein	ja	ja	ja	ja	ja	ja
IN BACKGROUND TASK	exactly once	ja	ja	nein	ja	nein	ja	nein	nein
IN UPDATE TASK	exactly once	nein	V1 / V2	nein	ja	nein	ja	nein	nein
STARTING NEW TASK	at least once	ja	nein	nein	ja	nein	ja	ja	ja
RECEIVE RESULTS FROM				ja	nein	nein	ja	ja	ja

Tabelle 4.1: Eigenschaften verschiedener RFC-Aufrufmodi

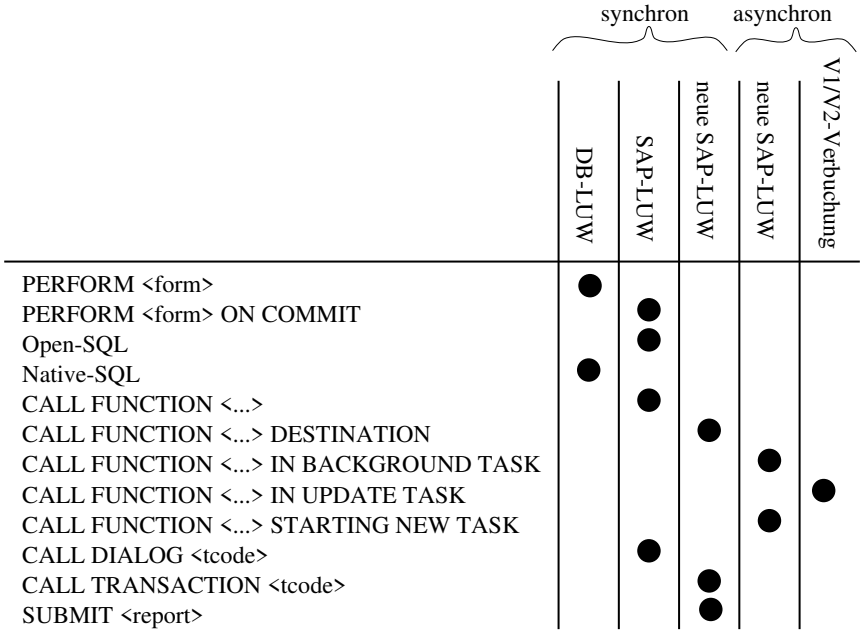


Tabelle 4.2: Zusammenhang der ABAP/4-Programmierung mit dem Transaktionensystem

4.2 Die RFC-Bibliothek des SAP R/3 für externe Systeme

In Abschnitt 3.1.3 des vorigen Kapitels wurde der Remote Function Call als offene integrative Schnittstelle der Applikationsschicht des SAP R/3 zu Fremdsystemen vorgestellt. Und in Abschnitt 4.1 sind die Funktionsbausteine als Kommunikationsendpunkte des Remote Function Call, deren Verwaltung und Definition, sowie die verteilte Programmierung innerhalb des R/3-Systems vorgestellt worden, deren Grundlage das System der Transaktionen in SAP R/3 ist (siehe Abschnitt 3.2.5).

Als Erweiterung der SAP-internen Client/Server-Kommunikation liefert SAP systemabhängige RFC-Bibliotheken[6] für die Kommunikation mit externen Systemen aus (→[RFC96a]). Die Funktionen dieser RFC-Bibliotheken bilden eine Programmierschnittstelle, um synchrone, asynchrone und transaktional asynchrone Aufrufe (→[BEG96]) von Funktionsbausteinen eines SAP-Applikationsserver durchzuführen. Desweiteren bietet sie die Möglichkeit selbst Service-Funktionen der externen Anwendung zu installieren, so daß eine bidirektionale Kommunikation zwischen dem SAP R/3 und der externen Anwendung möglich ist.

Im folgenden wird die verteilte Programmierung über das RFC-Protokoll für SAP-externe Systeme analysiert. Die Ziele dieses Abschnitts bestehen darin eine hinreichend genaue Beschreibung des Objektmodells der externen Repräsentation von SAP-Funktionen, den Service-Funktionen, zu erhalten und das dynamische Verhalten der Programmierschnittstelle für externe Systeme zu spezifizieren, so daß darauf aufbauend im nächsten Kapitel 5 ein Schichtenmodell der Gateway-Integration entwickelt werden kann.

4.2.1 Ein Modell für externe RFC-Funktionen

Einleitend wurde bereits auf die Möglichkeiten der RFC-Kommunikation hingewiesen, die verschiedenen Welten aus SAP R/3-, SAP R/2- und externen Systemen zu verbinden. Jedoch führen einige Einschränkungen des in Abbildung 4.6 dargestellten Objektmodells für externe Funktionen gegenüber dem für interne RFC-Aufrufe bei den Parameterarten und den Parametersignaturen zu einigen Einschränkungen.

Zunächst ist sind die vier Parameterarten der internen RFC-Kommunikation (vgl. Abschnitt 4.1.2) für externe RFC-Aufrufe unter Auslassung der Changing-Paramterart auf Eingabe-, Ausgabe- und Tabellen-Parameter (siehe Abbildung 4.6) beschränkt. Für die ersteren beiden fällt die Referenzoption weg, so daß immer eine „call by value“-Aufrufsemantik verwendet wird. Bei Tabellenparametern hingegen wird für externe RFC-Aufrufe, die den verteilten Aufruffall implizieren, immer die Kopiersemantik über den Delta Manager angewendet im Unterschied zur Referenzsemantik der internen RFC-Kommunikation.

Die Signaturen einzelner Parameter setzen sich, vergleichbar der internen Spezifikation (vgl. Abschnitt 4.1.2), aus Namen, Typen und Längenangaben zusammen. Jedoch sind als Typen für normale Parameter und Tabellenparameter weder Typdeklarationen aus Typen-Pools noch ABAP/4-Strukturen zugelassen, die im externen System garnicht verfügbar wären. Stattdessen verweisen die Signaturen direkt auf externe Typredefinitionen, die den ABAP/4-Typen entsprechen.

Bei Tabellen-Parametern sind im Unterschied zu Ein/Ausgabe-Parametern mehrfache Attribute entweder des Typs Character `C` oder Hex `X` möglich aber ebenso werden keine Typdeklarationen aus Typen-Pools oder heterogenen Strukturen unterstützt. Bei den Ein/Ausgabe-Parametern führt das nicht zu Problemen, weil eine Struktur einfach auf mehrere Parameter ausgefaltet werden kann. Bei Tabellenparametern hingegen führt ein äquivalentes Vorgehen zu mehreren unabhängigen Tabellenparametern mit der Konsequenz des Verlusts der ihrer Schlüsselsemantik.

Zur ihrer Definition ist für die Ein/Ausgabe-Parameter jeweils ein Array mit `RFC_PARAMETER`-Strukturen anzulegen, die bei dem Aufruf von `RfcCall()`, `RfcCallRecv()` und `RfcReceive()` Verwendung finden. Die Strukturen enthalten zunächst die Signatur aus Name, Typ und optionaler Länge[7].

[6] Für MS-Windows Umgebungen wird eine dynamische Link-Bibliothek ausgeliefert und für Unix-Umgebungen eine statische Link-Bibliothek.

[7] Bei einigen ABAP/4-Typen wird die Länge aus dem Typ inferiert.

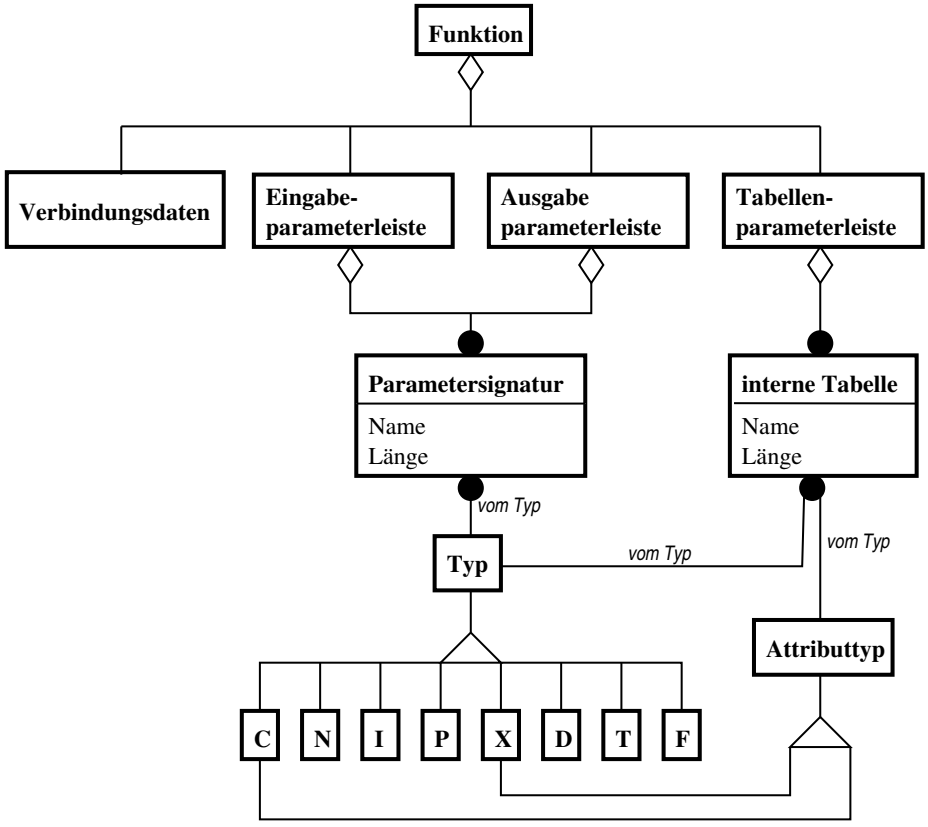


Abbildung 4.6: Objektmodell für externe RFC-Funktionen

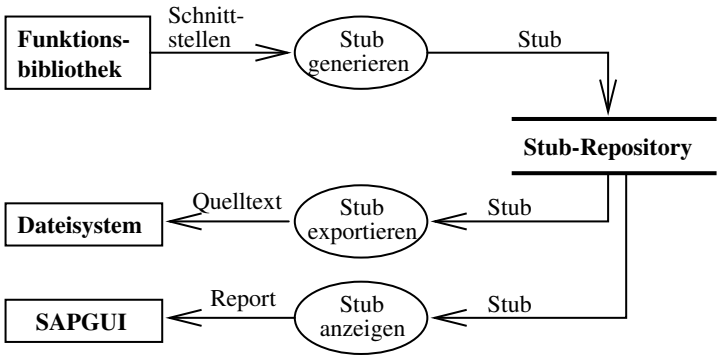


Abbildung 4.7: Programmgenerierung mittels des RFC-Generators

Desweitern besitzen sie ein Adreßfeld, welches explizit mit einer Referenz auf ein Speicherobjekt zur Aufnahme des Wertes belegt werden muß.

Tabellen-Parameter müssen bei aktiven Aufrufen mittels `itCreate()` erzeugt in die `RFC_TABLE`-Struktur eingetragen werden. Beim passiven Aufruf wird das von der RFC-Bibliothek erledigt, mit dem Effekt, daß immer die initiierende Seite die Tabellenparameter stellt.

4.2.2 Entwicklungsunterstützung durch die Workbench

Die Programmentwicklung für den Remote Function Call in MS-Visual-Basic oder die Programmiersprache C wird von der Workbench durch den *RFC-Generator* unterstützt (→[RFC96b]). Er ermöglicht die Erzeugung von Stub-Code für alle entfernt aufrufbaren Funktionsbausteine, der die Parameterkomposition und das dynamische Aufrufprotokoll abstrahiert.

In Abbildung 4.7 ist dargestellt, daß der RFC-Generator aus der Schnittstelle eines Funktionsbausteins den Stub-Code direkt in ein internes Repository schreibt. Von dort wird der Stub-Code entweder angezeigt oder als Datei exportiert. Nun kann ein externes Programm unter Verwendung der RFC-Bibliothek und dieser Stub-Schnittstellen erstellt werden, so daß sich der Entwicklungszyklus für ein externes Programm wie folgt darstellt:

1. Entwicklung der ABAP/4 Funktionsbausteine als Service-Funktionen für externe Client-Funktionen.
2. Erzeugen externer Stub-Schnittstellen und Testprogramme mittels des RFC-Generators.
3. Externes Ergänzen der Stub-Schnittstellen zu vollständigen Applikationen durch Hinzufügen von Quellcode.
4. Übersetzen und Binden der externen Programme.
5. Durchführen der Testsequenzen mittels der Testprogramme oder Ausführen der Applikationen.

Dabei kann Schritt Nummer drei übersprungen werden, wenn nur externe Testprogramme erzeugt, übersetzt und ausgeführt werden sollen.

Im folgenden wird die Kommunikation aus der Sicht des externen Systems beschrieben. Deshalb ist der Aufruf eines RFCs durch das externe Programm ein *aktiver Funktionsaufruf* im Unterschied zum *passiven Funktionsaufruf*, der es dem externen System gestattet Funktionen in einem Dämon-ähnlichen Prozeß (→[Ste92] Seite 79ff) dem SAP-System als Dienst zur Verfügung zu stellen.

4.2.3 Der aktive Funktionsaufruf

Der aktive Funktionsaufruf setzt voraus, daß die SAP-Zugangsberechtigung des anmeldenden Benutzers für einen Mandanten existiert, der wiederum berechtigt ist, die betreffenden Funktionsbausteine

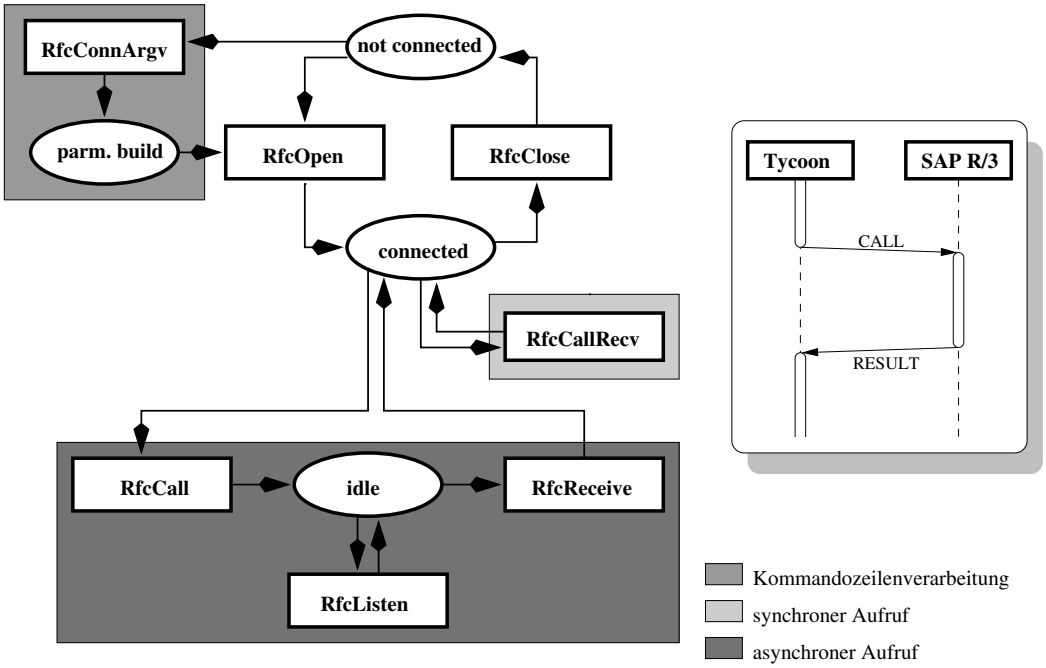


Abbildung 4.8: Zustandsmodell des RFC für einen aktiven Funktionsaufruf

aufzurufen[8]. Damit gestaltet sich das Szenario eines aktiven Funktionsbausteinaufrufs so, wie es in Abbildung 4.8 als dynamisches Modell dargestellt ist.

Ausgehend vom Zustand **not connected** muß zunächst die Verbindungsinformation, die in Abschnitt 4.2.3 beschrieben wird, als Speicherstruktur angelegt werden. Das wird entweder direkt durch das auführende Programm gemacht oder alternativ über den Aufruf der Funktion `RfcConnArgv()`, der aus einer übergebenden Kommandozeile entsprechende Strukturen generiert. Diese Verbindungsinformation wird der Funktion `RfcOpen()` als Parameter übergeben, die damit eine Verbindung zum entsprechenden Applikationsserver herstellt. Dann liefert die Funktion einen Handle zurück, das die Verbindung bei folgenden Aufrufen der RFC-Schnittstelle identifiziert, so daß mehrere Verbindungen nebenläufig aufgebaut werden können. Ein Fehler in der Autorisation gegenüber dem Applikationsservice, z.B. wegen eines falschen Passworts, wird beim Aufbau der Verbindung noch nicht erkannt.

Der Programmablauf ist danach in den Zustand **connected** überführt. Das Programm kann nun einen synchronen, einen asynchronen, einen transaktionalen asynchronen Aufruf oder die Beendigung der Verbindung mit `RfcClose()` durchführen.

Für einen Aufruf müssen, die in Abschnitt 4.1.1 beschriebenen Eingabe- und Ausgabeparameterstrukturen, sowie entsprechende Tabellenparameterstrukturen angelegt werden. Damit kann die Funkion `RfcCallRecv()` unter Angabe des auszuführenden Funktionsbausteins aufgerufen werden. Sie blockiert den Kontrollfluß des aufrufenden Programms bis entweder ein Fehler, eine Ausnahme, die Anforderung eines Rückrufs oder das Ergebnis des Aufrufs vorliegen und synchronisiert dadurch den Aufrufer mit der Ausführung des entfernten Funktionsbausteins.

Alternativ wird durch die Funktion `RfcCall()` ein asynchroner Aufruf eingeleitet. Der Kontrollfluß beim aufrufenden Programm läuft weiter. Mit dem Befehl `RfcListen()` kann daraufhin asynchron überprüft werden, ob bereits ein Ergebnis des Funktionsaufrufs vorliegt. Wenn das der Fall ist, dann wird durch die Funktion `RfcReceive()` festgestellt, ob ein Fehler, eine Ausnahme, die Anforderung

[8]Die Zugriffsberechtigungsprüfung in SAP erfolgt stehts durch explizite Befehle in ABAP/4. Tabellen des Systems unterliegen keinerlei Schutzmechanismus gegenüber dem Zugriff aus ABAP/4. Daher genügt bei Funktionsbausteinen, die selbst keine Berechtigungsprüfung durchführen, bereits der Systemzugang um beliebige Änderungen durchzuführen!

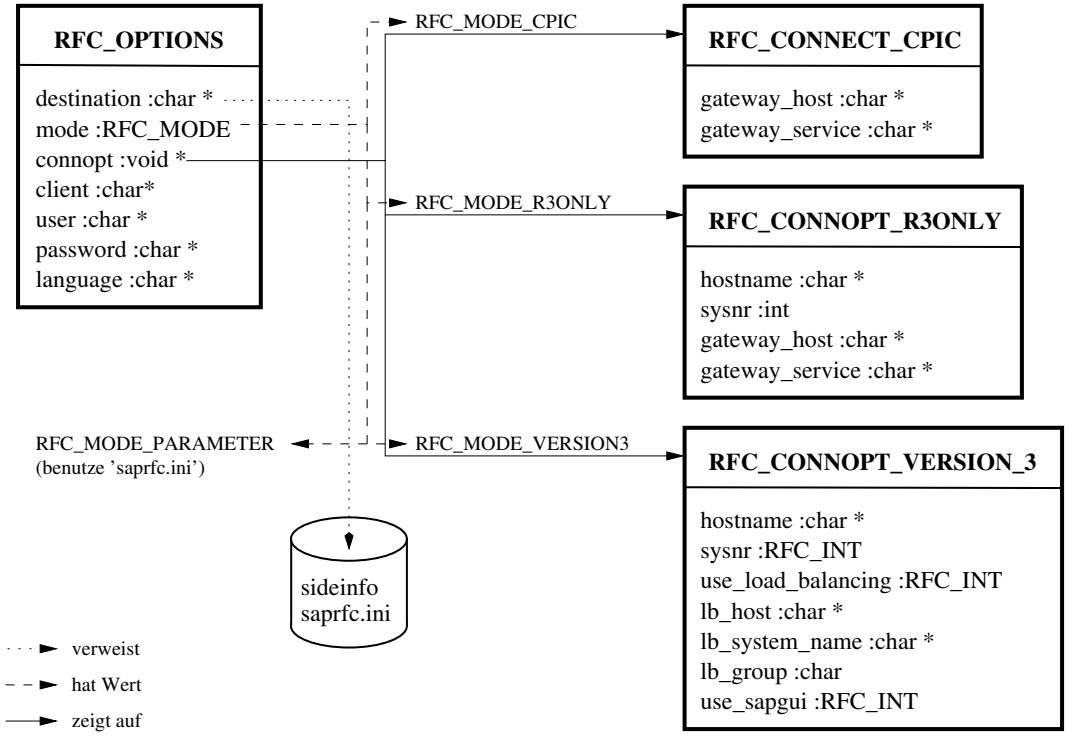


Abbildung 4.9: Logon-Information eines RFC-Aufrufs

eines Rückrufs oder das Ergebnis des Aufrufs vorliegt. Im Fall des Ergebnisses befindet sich das Programm wieder im Zustand **connected**. Weitere Aufrufe können dann durchgeführt werden.

In jedem Ausführungszustand der RFC-Schnittstelle, abgesehen von **not connected**, kann durch die Funktion `RfcAbort()` ein Abbruch der Verbindung durchgeführt werden. Dies kann differenziert für bestimmte Verbindungen durchgeführt werden oder durch Angabe eines null-Zeigers für alle Verbindungen der Schnittstelle gleichzeitig. Ein zweiter Parameter übernimmt eine Zeichenkette, die als Fehlertext der Gegenseite der Verbindung übermittelt wird und in ABAP/4 über das `MESSAGE`-Schlüsselwort (vgl. Abschnitt 4.1.2) empfangen werden kann..

Logon-Informationen

Zur Durchführung eines RFC-Aufrufs müssen der verbindungseröffnenden Funktion `RfcOpen()` Authentisierungsinformationen und die Applikationsserveridentifikation als `RFC_OPTIONS`-Struktur, wie in Abbildung 4.9, übergeben werden. Die Authentisierungsinformation besteht aus der Mandantennummer, dem Namen, dem Passwort und einem Sprachenschlüssel. Diese vier Angaben sind obligatorisch und einfach in die Struktur einzutragen.

Die Applikationsserveridentifikation ist davon abhängig, welche Version und welches Release der SAP-Software vorliegt. Häufige Änderungen der Protokolle und Eigenschaften haben bei SAP immer neue Mechanismen geprägt. Ausschlaggebend sind die beiden Felder `destination` und `mode` der Struktur `RFC_OPTIONS`. `destination` ist ein Verweis auf Einträge der Dateien `sideinfo` und `saprfc.ini`. `mode` steuert welches Protokoll mit Daten aus welcher Struktur versorgt wird (siehe Abbildung 4.9). Dabei sind folgende Werte für `mode` möglich:

`RFC_MODE_CPIC` Das R/2-Protokoll für SNA-Verbindungen über CPI-C wird ausgewählt. Die in der Struktur `RFC_CONNECT_CPIC` nicht vorhandenen Verbindungsdaten, werden vom Kommunikations-Gateway (vgl. Abschnitt 3.1.1) aus der `sideinfo`-Datei geholt. Dieses Protokoll funktioniert für

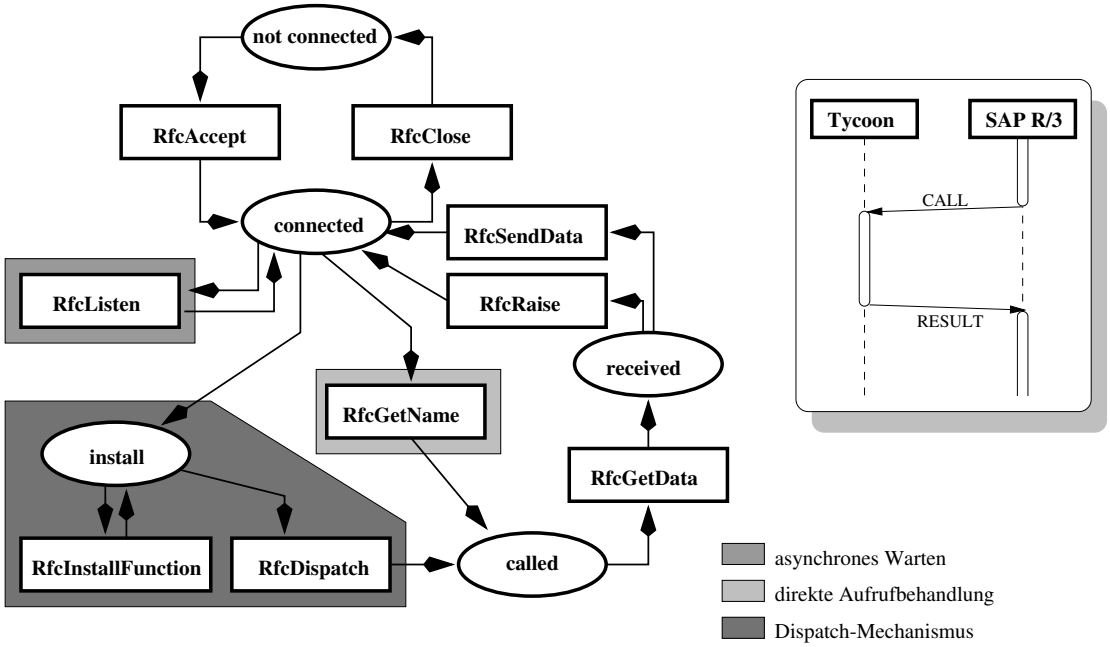


Abbildung 4.10: Zustandsmodell des RFC für einen passiven Funktionsaufruf

alle Verbindungstypen zwischen R/2, R/3 und externen Systemen.

`RFC_MODE_R3ONLY` Ein ausschließliches R/3-Protokoll wird eingestellt.

`RFC_MODE_VERSION3` Ein Protokoll für Verbindungen zu SAP R/3-Systemen ab Version 3.0c.

`RFC_MODE_PARAMETER` Allgemeinste Adressierungsvariante, die die vorigen drei einschließt. Die notwendigen Parameter werden aus der „saprfc.ini“-Datei gelesen.

4.2.4 Der passive Funktionsaufruf

Der passive RFC unterscheidet sich hinsichtlich seiner Ablaufsteuerung vom aktiven RFC. Ausgehend vom Zustand **not connected** wird zunächst, wie es in Abbildung 4.10 dargestellt ist, die Funktion `RfcAccept()` aufgerufen. Sie wirkt nicht blockierend auf den Kontrollfluß, erzeugt aber ein Handle, das im folgenden die Verbindung identifiziert. Daraufhin kann mittels der Funktion `RfcListen()`, ebenfalls nicht blockierend, festgestellt werden, ob ein eingehender Funktionsaufruf vorliegt.

In dem Zustand **connected** gibt es dann zwei alternative Strategien, eingehende Funktionsaufrufe zu bedienen. Die erste Alternative ist der Aufruf von `RfcGetName()`, wodurch der Name der aufzurufenden Funktion festgestellt wird. Das Server-Programm muß aus dem Namen die Funktion ermitteln und explizit aufrufen.

Die zweite Alternative ist ein „Dispatching“-Mechanismus, der die Registratur mehrerer Service-Funktionen des Prototyps

```
RFC_RC ( * RFC_ONCALL ) ( RFC_HANDLE handle );
```

mittels `RfcInstallFunction()` gestattet. Daraufhin kann der Aufruf der angeforderten Service-Funktion mit `RfcDispatch()` durchgeführt werden. Die letztere Strategie hat zwei Vorteile:

1. bei der Aufrufverteilung durch `RfcDispatch()` stehen folgende Service-Funktionen automatisch zur Verfügung:

`RFC_DOCU` liefert eine Liste aller registrierten Service-Funktionen

`RFC_PING` prüft die Verbindung

`RFC_SYSTEM_INFO` stellt einige Systemparameter fest

2. bei der Registratur wird für jede Service-Funktion ein dokumentierender String angegeben.

Allerdings hat diese Methode den Nachteil, der aufgerufenen Service-Funktion ein neues Verbindungs-Handle zu übergeben, was im Falle mehrfacher Rückrufe zu erheblichen Problemen führt (siehe folgender Abschnitt 4.2.5).

Ausgehend von dem Zustand **called** können dann die voralloziierten Parameterstrukturen der Service-Funktionen mit dem Aufruf `RfcGetData()` gefüllt werden. Nun kann auf den Parametern die eigentliche Service-Funktion ausgeführt werden. Wenn es vorzeitig zu einem Fehler kommt, kann mit der Funktion `RfcRaise()` eine Ausnahme ausgelöst werden. Andernfalls werden die Ergebnisparameter mit `RfcSend()` an den Aufrufer der Service-Funktion zurückgesandt und die Verbindung kann mit `RfcClose()` beendet werden.

Als Parameterwerte der Funktion `RfcAccept()` sind folgende Angaben möglich:

- Indirekte Angabe der Parameterwerte durch eine `saprfc.ini`-Datei:

  `-D<name>` Name eines Eintrags in einer `saprfc.ini`-Datei

  `-t` Debug-Modus

- Direkte Angabe der Parameterwerte:

  `-a<program ID>` Eindeutige Programmidentifikation

  `-g<gwhostname>` Host-Name des Gateway-Handlers

  `-s<gwservice>` Prozeßname des Gateway-Handlers

  `-t` Debug-Modus

In diesem Zusammenhang muß angemerkt werden, daß die passive Kommunikation mit dem SAP R/3-Applikationsserver bis zur Version 3.0b ausschließlich durch das explizite Starten des Server-Programms mit den Service-Funktionen durch das SAPGUI oder einen Gateway-Handler über das UNIX `rsh`-Programm möglich war. Ab Version 3.0c ist zusätzlich die Regstrierung des Service-Programms bei einem Gateway-Handler möglich, so daß nicht jeder erneute Funktionsaufruf den Start eines Prozesses erfordert.

4.2.5 Rückrufe in Funktionsaufrufen

Bei Rückrufen aus dem SAP-System nach einem aktiven Funktionsaufruf oder Aufrufen von Funktionsbausteinen innerhalb eines passiven Funktionsaufrufs entstehen einige Probleme mit der Verwaltung der Kommunikations-Handles und Parameter.

Der dynamische Ablauf dieses Szenarios in Abbildung 4.11 ergibt sich aus den dynamischen Modellen für den passiven und aktiven Funktionsaufruf (siehe Abbildungen 4.8, 4.10). Durch Vereinigung der Zustände **connected** und **received** erhält man eine adäquate Darstellung des Rollenwechsel im externen System vom Aufrufer beim aktiven Funktionsaufruf zum Diensterbringer beim passiven Funktionsaufruf.

In der Rolle des Aufrufers wird der initiierende Prozeß des externen Systems mit `RfcCallRecv()` oder `RfcReceive()` das Ergebnis eines Aufrufs entgegennehmen. Der Rückgabewert dieser Funktionen zeigt[9] dann einen abgeschlossenen Funktionsaufruf (`RFC_OK`) oder einen eingeleiteten Rückruf (`RFC_CALL`) durch das SAP-System an. Daraufhin wird in den Zustand **callback** gewechselt, der zwingend eine Behandlung des Rückrufs durch `RfcDispatch()` oder `RfcGetName()` erfordert.

[9] Weitere Fälle, wie das Auftreten einer Ausnahme, werden hier nicht betrachtet.

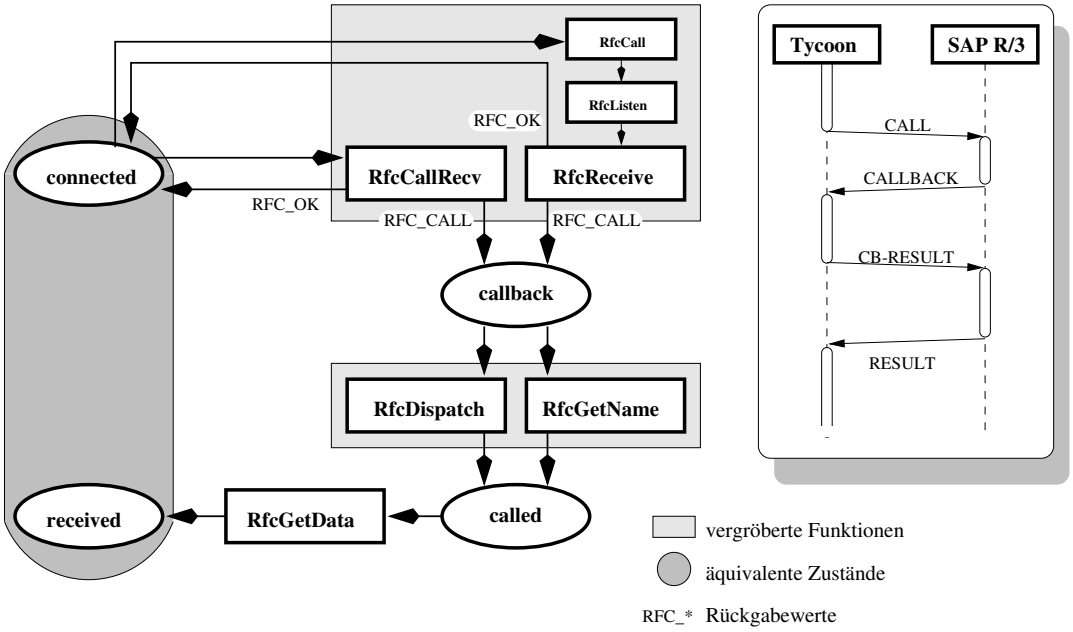


Abbildung 4.11: Zustandsübergänge bei einem RFC-Rückruf

Im anderen Fall der Rolle des Diensterbringers wird nach einem eingehenden Funktionsaufruf in den Zustand **received** gewechselt. Dieser entspricht der Abarbeitung der aufgerufenen Funktion und somit durch die Möglichkeit eines Rückrufs auch dem Zustand **connected** des aktiven Funktionsaufrufs.

Durch mehrfachen Kontrollflußwechsel zwischen dem externen System und SAP R/3 gelangt man zu einem zyklischen Durchlaufen des dynamischen Aufrufmodells aus Abbildung 4.11. Hierbei sind zwei Probleme gesondert zu betrachten:

1. Jeder Aufruf der Schnittstellenfunktion `RfcDispatch()` erzeugt ein neues Kommunikations-Handle. Das führt bei wiederholtem Aufruferwechsel bei Anwendung der Rückruf-Funktionalität zu mehreren Handles.

2. Ein mehrfacher Aufruf derselben Funktion im externen System erfordert die simultane Verwaltung entsprechender Parameterstrukturen (→ Abschnitt 4.2.1).

Deshalb sind bei der Implementation einer externen RFC-Schnittstelle Stack-ähnliche Konstruktionen zur Verwaltung der Kommunikations-Handles und Parameterstrukturen vorzusehen, deren Stacktiefe der jeweiligen Rückrufverschachtelung entspricht.

4.2.6 Besonderheiten beim transaktionalen RFC

Wie für die interne SAP R/3-Kommunikation gibt es für die Kommunikation mit externen Systemen die Möglichkeit Funktionsaufrufe *transaktional* mit einer „exactly once“-Aufrufsemantik durchzuführen. Jedoch sind hier einige besondere Vorkehrungen bzgl. der Aufrufsemantik zu treffen, die innerhalb des SAP R/3 automatisch vom Laufzeitsystem durchgeführt werden.

Konkret kann ein transaktional durchgeführter RFC als verteilte Transaktion im Sinn der Datenbanktheorie angesehen werden. Dann sind Atomarität und Dauerhaftigkeit mit der „exactly once“-Aufrufsemantik gleichzusetzen. Diese Eigenschaften werden durch ein zwei-Phasen-Commit-Protokoll (→[CDK94, Reu87]) sichergestellt:

Phase 1: Funktionsübertragung Alle RFC-Funktionsaufrufe mit ihren Parametern und einem eindeutigen Transaktionsidentifikator werden zu zum Diensterbringer übertragen.

Der Diensterbringer überprüft anhand der Transaktions-ID ob die RFC-Funktionsaufrufe alle korrekt durchgeführt wurden.

Der Diensterbringer teilt dem Aufrufer die erfolgreiche Durchführung der RFC-Funktionen mit.

Phase 2: Bestätigung Der Aufrufer teilt dem Diensterbringer die Durchführung mit, woraufhin beide Systeme den Transaktions-ID löschen können.

Die Abwicklung dieses Protokolls kann aus zwei Gründen nicht vollständig in der RFC-Bibliothek verborgen werden:

- Die zur persistenten Speicherung der Tids notwendigen Dienste, wie z.B. eine Datenbank, sind nicht in einheitlicher Form in jeder Systemumgebung vorhanden.
- Die RFC-Bibliothek kann von sich aus Netzwerkübertragungen nicht aus beliebigen Gründen zu beliebiegen Zeitpunkten wiederholen.

Deshalb sind zur Durchführung eines transaktionalen RFCs erstens Vorkehrungen zur persistenten Speicherung der Transaktions-ID zu treffen, das kann notfalls durch eine Datei realisiert werden, und zweitens muß ein Mechanismus vorgesehen werden, der transaktionale Funktionsaufrufe, die durch einen System- oder Netzwerkfehler abgebrochen wurden, wieder aufnimmt und weiterbearbeitet.

Aktiver transaktionaler RFC

Beim aktiven transaktionalen RFC muß nach dem Einrichten der Verbindung mit `RfcOpen()` vom SAP-System mit `RfcCreateTransID()` eine Transaktion-ID angefordert werden. Diese ist lokal, persistent zu speichern und wird im transaktionalen Aufruf einer Service-Funktion durch `RfcIndirectCall()` verwendet. Wenn es bei dem Aufruf durch `RfcIndirectCall()` einen Fehler gibt oder sogar der Systemausfall eintritt, muß der Aufruf mit derselben Transaktions-ID zu einem späteren Zeitpunkt wiederholt werden. Der Aufruf wird mit `RfcClose()` abgeschlossen und ist auf die Ausführung einer Service-Funktion beschränkt.

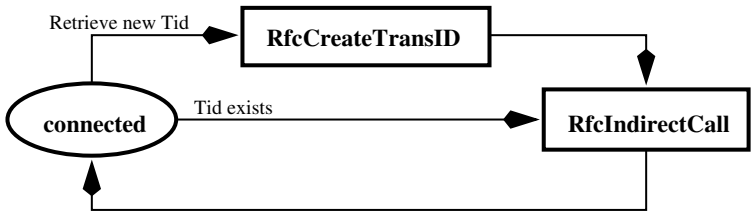


Abbildung 4.12: Transaktionaler Aufruf einer Funktion

Die vorstehende Abbildung 4.12 ergänzt den dynamischen Ablauf des aktiven Funktionsaufrufs (siehe Abbildung 4.8). Zusätzlich sind aber die erwähnten Vorkehrungen zu treffen, die bei einem Abbruch durch Netzwerkfehler oder einen Systemausfall zu einem späteren Zeitpunkt den Funktionsaufruf mit demselben Tranaktions-ID wiederholen, bis die Funktion erfolgreich durchgeführt wurde.

Passiver transaktionaler RFC

Der passive Transaktionale RFC gestaltet sich durch die Realisierung der Commit- und Rollback-Funktionalitäten erheblich schwieriger. Es sind nicht nur die Tids zu speichern und im Fehlerfall der erneute Funktionsaufruf zu realisieren, sondern es müssen zusätzliche Funktionen zur Realisierung der beiden Kommunikationsphasen *Funktionsübertragung* und *Bestätigung* programmiert werden. Dazu dienen folgende vier Callback-Funktionen:

onCheckTid Empfang einer Transaktions-ID. Es wird getestet, ob diese zu einem laufenden Aufruf gehört. Wenn ja, dann muß das mit einem Rückgabewert ungleich null signalisiert werden. Wenn nicht, dann muß die Transaktions-ID persistent gespeichert und dies durch eine null als Rückgabewert signalisiert werden.

onCommit Alle Funktionen sind aus der Sicht der RFC-Bibliothek erfolgreich aufgerufen worden. Nun müssen die Ergebnisse durch ein Commit der Datenbank oder ähnliches externalisiert werden.

onRollback Von der RFC-Bibliothek ist ein Fehler erkannt worden, der zum Abbruch der lokalen Verarbeitung führt. Zwischenergebnisse werden ungültig. Bei der Arbeit auf einer Datenbank ist ein Rollback einzuleiten.

onConfirmTid Die Verarbeitung des transaktionalen RFC ist aus der Sicht des SAP-Systems erfolgreich durchgeführt worden. Das aufrufende SAP R/3-System signalisiert hiermit, das der zugehörige Transaktions-ID ungültig geworden ist und gelöscht werden kann.

Diese vier Funktionen werden mit `RfcInstallTransactionControl()` bei der RFC-Bibliothek registriert, bevor in die eigentliche Verarbeitung der eingehenden Funktionsaufrufe mit `RfcDispatch()` verzweigt wird. Deshalb muß für den passiven Transaktionalen RFC der dynamische Ablauf entsprechend Abbildung 4.10) ergänzt werden.

Abbildung 4.13: Installation der Hilfsfunktionen für die Transaktionsabwicklung

4.3 Zusammenfassung

Die Funktionsbausteine sind die Grundlage für Verteilung, Verbuchung, Nebenläufigkeit und eine offene Schnittstelle des SAP R/3. Sie sind die Kommunikationsendpunkte für RFC-Aufrufe durch eingebettete ABAP/4-Anweisungen und von externen Systemen. Funktionsbausteine haben eine in der Funktionsbibliothek festgelegte Schnittstelle mit verschiedenen Parameterarten und Ausnahmen. Das Aufrufprotokoll des RFC unterstützt asynchrone, synchrone und transaktionale Aufrufsemantiken mit jeweils davon abhängigen Möglichkeiten für Parameter, Ausnahmen und die Rückruffunktionalität.

Durch die RFC-Bibliothek für externe Systeme werden Funktionsbausteine zur offenen Schnittstelle des SAP R/3. Sie ermöglicht die bidirektionale RFC-Kommunikation mit einem SAP-Applikationsserver. Ihr Objektmodell ist jedoch in den Parameterarten- und Typen, speziell bei Tabellenparametern, eingeschränkt gegenüber der in ABAP/4 eingebetteten Aufrufschnittstelle.

Die Beschreibung der Logon-Information zur Netzwerkadressierung, der externen Parameterrepräsentation und der dynamischen Abläufe an der Schnittstelle beim aktiven und passiven Aufruf und im Fall eines Rückrufs sind die Vorraussetzung für die Verwendung der RFC-Bibliothek als Integrationsschnittstelle im folgenden Kapitel 5.

Kapitel 5

Konzepte zur Integration des SAP R/3 in Tycoon

In Kapitel 2 wurden die wichtigsten Integrationsschnittstellen der Datenbankprogrammiersprache Tycoon untersucht. Aus einer Betrachtung bestehender Einbindungen von Fremdsystemen in die Sprachumgebung Tycoon in Abschnitt 2.3.5 wurde die Schnittstelle für dynamisches Binden externer C-Funktionen als dominierender Basismechanismus der Integration identifiziert. Außerdem ergaben sich Applikationstransparenz, Typsicherheit, Generizität und Datenunabhängigkeit als wichtige Eigenschaften der integrativen Einbindung.

Weiterhin ist in Kapitel 3 das System SAP R/3 unter dem Gesichtspunkt der integrativen Offenheit (siehe Abschnitt 3.1.3) analysiert worden. Es wurden in Abschnitt 3.1.2 die semantischen Objektklassen SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen identifiziert. Der Remote Function Call bietet als offene Schnittstelle weitreichende Möglichkeiten zum Zugriff und der Steuerung der semantischen Objekte des SAP R/3 (siehe Abschnitt 3.2.4 bis 3.2.4). Deshalb folgte eine genaue Untersuchung der Client/Server-Programmierung auf Basis des RFC für ABAP/4-Programme (vgl. Abschnitt 4) und für SAP-externe Systeme (vgl. Abschnitt 4.2).

Damit sind alle Voraussetzungen zur Integration des SAP R/3-Systems in die Programmierwerkbank Tycoon-Tl über eine Gateway-Schnittstelle auf der Basis des RFC geschaffen. Im vorliegenden Kapitel werden die Anforderungen an eine solche Gateway-Integration formuliert und spezielle Probleme, wie die automatengesteuerte Behandlung flüchtiger Resourcen, Mobilität und Persistenz der Objekte innerhalb von Tycoon und Performanz in der Ausführung, sowie deren Lösungen vorgestellt. Danach wird die vierschichtige Software-Architektur der prototypischen Gateway-Implementation im Rahmen dieser Arbeit dargestellt. Sie basiert auf der RFC-Programmierbibliothek für SAP-externe Systeme und abstrahiert C-Bindungen, die notwendig sind, um die Anbindung der RFC-API-Funktionen in den programmiersprachlichen Raum Tycoon zu schaffen. Auf einer höheren Schicht werden automorphe Werte eingesetzt, um ein polymorphes Marshalling der Aktualparameter eines RFC-Funktionsaufrufs unter Verzicht auf generierte Stub-Schnittstellen zu realisieren (vgl. Abschnitt 2.3.6).

5.1 Vergleich der Systeme und Anforderungen an eine Tycoon-SAP R/3-Schnittstelle

Das Ziel der Integrationsarchitektur des SAP R/3-Tycoon-Gateways ist die Abbildung der R/3-Applikationen und Daten in den programmiersprachlichen Raum der Sprache Tl. Fundamentale Unterschiede der Systeme R/3 und Tycoon-Tl bezüglich ihrer Paradigmen, semantischen Objektklassen, Abstraktionen und Schnittstellen zu externen Systemen erfordern Mechanismen zu deren Anpassung bzw. Abbildung.

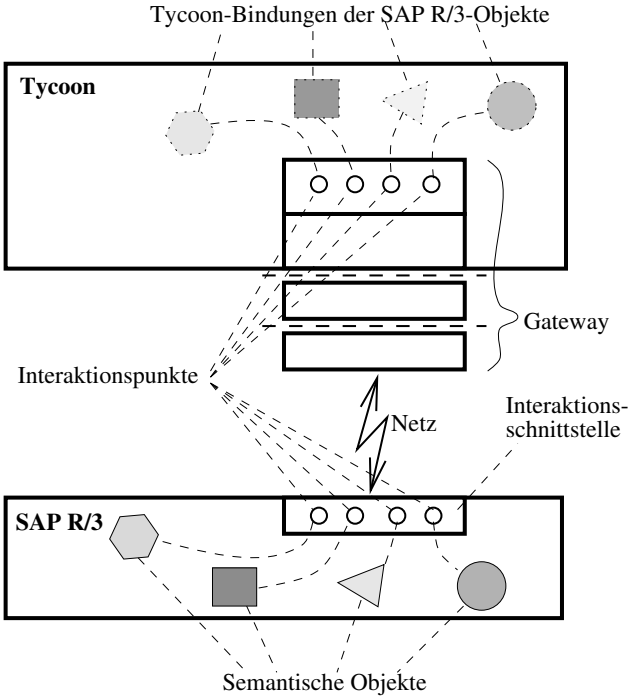


Abbildung 5.1: Architektur der Tycoon-SAP R/3-Integrationsschnittstelle

Wesentliches Merkmal des SAP R/3 ist die Methode „built in“, das heißt alle Mechanismen in die Sprache oder Systemumgebung zu integrieren und damit die Freiheitsgrade der Modellierung statisch festzulegen. Tycoon-Tl hingegen verfolgt einen „add on“-Ansatz mit bibliotheksbasierten Erweiterungen. Diese Paradigmen setzen einen Realisierungsspielraum bei der Evolution der Systeme bzgl. Erweiterungen und Anpassungen. Für die Integration des SAP R/3 in das Tycoon-Tl-System ist die statische Modellierungsmächtigkeit des SAP R/3 von Vorteil, weil den Umfang der Abzubildenden Objekte und Konstrukte beschränkt.

Alle R/3-Applikationen sind unter Verwendung der semantischen Objekte SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen in ABAP/4 formuliert. Dabei sind SAP-Transaktionen um Interaktionssemantik erweiterte ABAP/4-Programme (vgl. Abschnitte 3.1.2, 3.2.5). Tycoon-Tl bietet hingegen Daten, Code und Threads, wobei Threads um Laufzeitsemantik erweiterter Code sind. Die genannten semantischen Objekte beider Systeme können nahezu direkt den abstrakten semantischen Stufen (vgl. Abschnitte 2.1.1, 3.1.2)

Daten Daten stellen Literale, Werte oder Objekte dar, die zwar zustandsbehaftet sein können, aber keine Berechnugsvorschriften enthalten.

Funktionen Funktionen sind Daten, die interpretierbare Berechnugsvorschriften enthalten.

Abläufe Abläufe sind Funktionen, deren Berechnugsablauf explizit zustandsbehaftet ist und eventuell gesteuert werden kann.

zugeordnet werden. Eine Ausnahme der direkten Zuordnung bilden die SAP-Transaktionen, die keine Abläufe im obigen Sinn vergleichbar den Tycoon-Threads darstellen. Vielmehr sind sie Funktionen, die sich aus einer Folge aus Berechnungsvorschriften, nämlich den Ablauflogiken der Dynpros, zusammensetzen und bei ihrer Berechnung Seiteneffekte in dem SAP-System auslösen. Im Unterschied zu ABAP/4-Programmen werden sie nicht durch den ABAP/4-Prozessor, sondern durch den

Dynpro-Prozessor des Applikationsinstanz ausgeführt (vgl. Abschnitt 3.1.2). Sie können ihren Berechnungszustand aber weder selbst steuern, noch ist die aufrufende SAP-Transaktion oder das aufrufende ABAP/4-Programm dazu in der Lage. Lediglich ein Benutzer kann über Dynpro-Eingaben den Berechnugszustand respektive Ablauf beeinflussen, was allerdings als Teil der Berechnung angesehen werden muß. Entsprechend der Forderung nach Applikationstransparenz sollten also zumindest funktionale Objekte (ABAP/4-Programme, SAP-Transaktionen) durch das Gateway abgebildet werden.

Abstraktionen[1] des SAP R/3 sind faktisch nur durch Funktionsbausteine (vgl. Abschnitt 4) gegeben. Für Sperren(Isolation) gibt es explizite durch Sperrobjekte realisierte Funktionsbausteine, die eine entsprechende Leistungen kapseln aber auch explizit aufgerufen werden müssen. Zugriffsrechte werden explizit über ein „built in"-Kommando geprüft und Ausnahmen können ausschließlich über eine Systemvariable abgefangen werden. Tycoon-Tl hingegen bietet Funktionsabstraktionen (vgl. Abschnitt 2.1), Persistenz- und Mobilitätsabstraktionen und eine Abstraktion für die Ausnahmebehandlung. Von diesen Abstraktionen profitieren externe durch Gateways integrierte Anwendungen automatisch und erfahren damit eine Leistungserweiterung. Für die Gateway-Konstruktion bedingen sie dagegen erhöhten Aufwand, weil sie z.T. explizite Unterstützung erforderlich machen (vgl. Abschnitt 2.1.2, 2.1.3).

In Abschnitt 3.2.4 wurde gezeigt, daß die RFC-Schnittstelle des SAP R/3-Systems hinreichend offen ist, um indirekt über Funktionsbausteine vollständige Kontrolle über die semantischen Objekte des SAP R/3 und damit implizit auch die R/3-Applikationen zu erlangen. Andere Interaktionsschnittstellen des SAP R/3 sind entweder nicht verfügbar (vgl. Abschnitt 6.3) oder bieten zuwenig Offenheit (vgl. Abschnitt 3.1.3). Andererseits wurde oben bereits dafür argumentiert, Funktionsbausteine aus SAP R/3 auf Funktionen in Tycoon-Tl abzubilden.

Im folgenden wird deshalb von der in Abbildung 5.1 angedeuteten Gateway-Architektur auf der Basis von Funktionsbausteinen als Interaktionspunkte ausgegangen. Diese Interaktionspunkte dienen zum Zugriff auf SAP R/3-Objekte (vgl. Seite 50ff.) und können über einen Gateway-Mechanismus an Tl-Funktionsobjekte gebunden werden. Mit derartigen Funktionsobjekten der Sprache Tl können dann neue Applikationen unter Ausnutzung wichtiger Tycoon-Eigenschaften, wie Persistenz, Mobilität, Skalierbarkeit und Offenheit konstruiert werden.

Besondere Berücksichtigung müssen dabei die vier Eigenschaften Applikationstransparenz, Typsicherheit, Generizität und Datenunabhängigkeit erfahren, die das Ergebnis der Analyse existierender Gateway-Schnittstellen in Tycoon-Tl sind (vgl. Abschnitt 2.3.6).

Technische Anforderungen

Grundsätzlich muß für die Realisierung der Bindungen zwischen den beiden Alternativen der *dynamischen Bindung* und der *Stub-Code-Generation* ausgewählt werden (vgl. Abschnitt 2.3.6). Letztere empfiehlt sich für Schnittstellen mit einem statisch großen Umfang an Schnittstelleninteraktionspunkten, wie z.B. im Fall der Starview-Einbindung, die auf umfangreichen C++-Headerdateien basiert. Damit ist durch die C++-Headerdateien eine gemeinsame statische Spezifikation der Interaktionsschnittstelle gegeben. Desweiteren werden die Marshalling-Prozeduren statisch typisiert und optimiert.

Bei SAP R/3 liegen die Voraussetzungen jedoch anders. Zwar besitzt jede SAP R/3-Instanz einen umfangreichen Bestand an Funktionsbausteinen, die aber längst nicht alle für einen externen RFC-Aufruf vorbereitet sind. Zudem ist eine hohe Frequenz für Änderung der Funktionsbausteine durch Hinzufügen neuer bzw. Abändern bestehender Funktionsbausteine zu erwarten, so daß für das SAP R/3-Tycoon-Gateway die dynamische Bindung bevorzugt wird.

Aus den Voraussetzungen der SAP R/3-RFC-Schnittstelle mit zugehöriger API der RFC-Bibliothek für externe Systeme und der Tycoon-C-Calls-Schnittstelle ergeben sich damit folgende im Rahmen der Gateway-Architektur zu lösende Teilprobleme:

- Semantisch korrekte Integration der Funktionsbausteine durch Abbildung von
  - synchronem- und asynchronem Aufruf mit zugehöriger Kontrollflußsteuerung

[1] Eine wesentliche Leistung des SAP R/3 liegt in den Abstraktionen auf der Anwendungsebene, die nicht Teil dieser Arbeit sind.

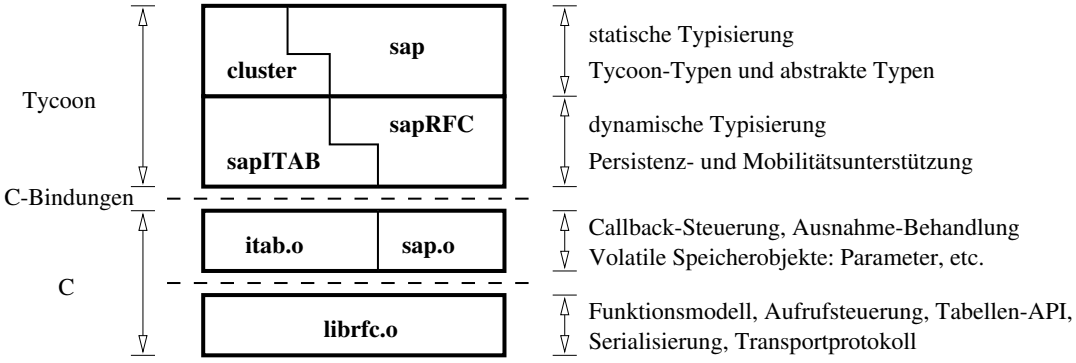


Abbildung 5.2: Vier-Schichtenmodell des Gateways

  - Eingabe- und Ausgabeparametermarshalling
  - Tabellenparameter
  - Ausnahmemechanismen

- Berücksichtigung von Zeit und Raumschranken:
  - Persistente Addressierung der Funktionsbausteine
  - Mobilität der Bindung an Funktionsbausteine
- Typsicherheit der abgebildeten Funktionen als eine der vier Gateway-Eigenschaften aus Kapitel 2.
- Performanz der Bindung und des Aufrufs.
- Laufzeitfunktionsbindung, wie oben Argumentiert.
- Aktiver und passiver Funktionsaufruf, damit bidirektionale Aufrufe ermöglicht werden.
- Portabilität der Schnittstelle. Insbesondere sollen die notwendigen Änderungen bei einer Portierung auf wenige umgebungsabhängige Funktionalitäten beschränkt sein.

Insbesondere aus dem letzten Punkt der Portabilität ergibt sich faßt automatisch eine Systemschichtung ($\rightarrow$[LD87, Nut92]) als Metamodell für die Gateway-Architektur, die die notwendigen Änderungen einer Portierung im Idealfall auf eine Gateway-Schicht begrenzen würde.

5.2 Softwarearchitektur des Integrations-Gateways

Aus den technischen Anforderungen des letzten Abschnitts und Abbildung 5.1 Seite 80 ist bereits ein grober Rahmen für die Architektur der SAP R/3-Tycoon-Systemintegration vorgegeben. Die offengelassene Spezifizierung des Gateways ist in Abbildung 5.2 wiedergegeben. Es setzt sich aus vier Schichten zusammen:

1. Die unterste Schicht eins enthält die RFC-Programmierbibliothek für C-Programme und ist mit ihren Mechanismen vollständig in Abschnitt 4.2 beschrieben. Sichtbare Konzepte dieser Schicht sind ihr Funktionsmodell (vgl. Abbildung 4.6), die Aufrufsteuerung der RFC-API und die Tabellen-API. Verdeckte interne Dienste umfassen die Parameterserialisierung und das Transportprotokoll. Im Fall einer Portierung würde sich diese von SAP vorgegebene Schicht ändern.

2. Die zweite Schicht ist eine Zwischenschicht basierend auf der Strategie der Zugriffsfunktionen (vgl. Abschnitt 2.2.2). Sie bietet Dienstfunktionen zur Erzeugung und Kontrolle komplexer Parameterstrukturen für die RFC-API der Schicht eins an. Desweiteren enthält sie Mechanismen der Rückrufsteuerung, der Ausnahme-Behandlung, zur Kontrolle der flüchtigen Speicherobjekte und ein gegenüber dem Funktionsmodell der RFC-API um Strukturen für Tabellen-Parameter erweitertes Funktionsmodell.

   Diese Schicht zwei wäre als Abstraktion der Schicht eins Gegenstand explizierter Änderungen für eine Portierung.

3. Die dritte Schicht bindet die Dienste der zweiten Schicht an Tycoon-Tl-Funktionen. Dazu enthält sie die beiden Module `sapITAB` für Tabellenparameter und `sapRFC` für das Funktionsmodell mit R/3-Server-Adressierung und Aufrufsteuerung. Ihr wichtigster Dienst ist die polymorphe `bind`-Funktion zur Bindung von Funktionsbausteinen. Funktionsparameter werden beim Aufruf gemäß der direkten Bindungstechnik (vgl. Abschnitt 2.3.6) als automorphe Werte zur dynamischen Typinspektion übergeben. Beide Module unterstützen Persistenz und Mobilität der Bindungen an Funktionsbausteine in Zusammenarbeit mit der Zwischenschicht zwei durch eine automatengesteuerte Verwaltung der flüchtigen Speicherobjekte.

4. Die vierte Schicht enthält als Abstraktion automorpher Parameterwerte komplexe Sequenzen zur deren Aggregation und Dekomposition. Dementsprechend wird ein vereinfachtes an gewöhnliche Tycoon-Tl-Funktionen angelehntes Modell für Funktionsbausteine angeboten. Alle Typprüfungen für Dienstfunktionen dieser Schicht sind statisch und kommen deshalb ohne automorphe Werte an der Schnittstelle aus.

Die folgende Beschreibung des Gateways ist nicht streng an den vier Schichten orientiert. Teilweise schichtübergreifende Mechanismen vor allem der Schichten zwei und drei würden ein derartiges Vorgehen verkomplizieren. Stattdessen werden bestimmte Problemstellungen, wie die Typabbildung, die Realisierung der Persistenzabstraktion oder die Ausnahmebehandlung, aufgegriffen und die Mechanismen zu ihrer Lösung im Zusammenhang dargestellt. Der Schicht vier, die keine solchen zur Schicht drei übergreifenden Mechanismen besitzt, ist der eigene Abschnitt 5.3 gewidmet.

Abzubildende dynamische Abläufe

Ausgehend vom Modell der RFC-Funktionen für externe R/3-Anbindungen müssen folgende Schritte zum einmaligen Aufruf eines Funktionsbausteins abhängig vom passiven oder aktiven Aufrufmodus durchgeführt werden (vgl. Abbildungen 4.8, 4.10 Seite 72ff):

Aktiv

1. Auswahl eines Applikationsserveradressierungsverfahrens und Erzeugen entsprechender Adreßstrukturen.
2. Aufbau einer Verbindung zum Applikationsserver mit der Adreßstruktur.
3. Aufbau der Parameterstrukturen.
4. Anlegen der Tabellenparameter über die Tabellen-API.
5. Aufruf des selektierten Funktionsbausteins mit den Parameterstrukturen und und Tabellenparametern.
6. Überprüfen des Aufrufergebnisses auf Auftreten einer Ausnahme oder eines Callbacks.
7. Auslesen der Ergebnisparameter und der Tabellenparameter.
8. Beenden der Verbindung.

Passiv

1. Auswahl zwischen dem `rsh`-Verfahren oder der Server-Registratur beim Gateway-Handler (vgl. Abschnitt 4.2.4.
2. Start des externen Programms bzw. Akzeptieren eines eingehenden Funktionsaufrufs.
3. Aufbau der Parameterstrukturen.
4. Aufruf des selektierten Funktionsbausteins mit den Parameterstrukturen und und Tabellenparametern.
5. Eventuell Auslösen einer Ausnahme oder eines Callbacks
6. Zurücksenden des Funktionsergebnisses.
7. Beenden der Verbindung.

Daraus ergeben sich Parameterbindungen, die dynamische Aufrufsteuerung und die Ausnahmebehandlung als Teilaspekte eines RFC-Aufrufs. Nachfolgend werden sie einzeln erläutert und in diesem Zusammenhang die Applikationschnittstellen der Schicht drei vorgestellt.

5.2.1 Parameter und ihre Bindungen

Die Aufgabe der Parameterbindung des Gateways liegt in der typkorrekten, effizienten Bindung von Werten des Aufruferkontextes an den Kontext der aufgerufenen Funktion. Dabei besitzt die aufgerufene Funktion eine Signatur, bestehend aus Namen, Typen und evtl. einer Längenangabe für die Parameter. In der typsicheren Tycoon-Tl-Sprache wird zu jedem Aktualparameter entsprechend dem Subtyppolymorphismus geprüft, ob sein inferierter Typ wenigstens die Spezifikation der in der Funktionssignatur angegebenen Typen erfüllt. Für die Tl-Funktionen auf SAP R/3-Funktionsbausteine abbildende Gateway-Konstruktion kommen weitere Probleme durch unterschiedliche Typsysteme, verschiedenartige Abstraktionen durch Funktionen bzw. Funktionsbausteine und systembedingt getrennte Ausführungskontexte hinzu.

Um also die Abbildung herzustellen, muß erstens eine Abbildung der in den Signaturen auftretenen Typen definiert werden und zweitens muß eine Möglichkeit gefunden werden, die Funktionsbausteine mit ihren verschiedenen Parameterarten an Tl-Funktionen zu binden, was durch die Schnittstelle der Gateway-Schicht drei realisiert ist (vgl. Abschnitt 5.2.3).

Abbildung der Typen aus SAP R/3 nach Tycoon

Bei einem Vergleich der Typsysteme aus Tl (vgl. Abschnitt 2.1) und aus SAP R/3 (vgl. Abschnitt 3.2.2) stellt man fest, daß das von Tl, wegen Eigenschaften, wie Polymorphismus, struktureller Subtypäquivalenz, Typoperatoren etc., mehr Flexibilität bietet, als das monomorphe Typsystem der Sprache ABAP/4 mit seinen acht Typen. Da auch die RFC-Programmierbibliothek in Gateway-Schicht eins im wesentlichen Abbildungen dieser ABAP/4-Typen und nicht der aufgesetzen externen Typen

SAP-ext-type	ABAP/4	RFC	C-Typ	Tycoon
CHAR	C	RFC_CHAR	char	:Char
INT4	I	RFC_INT	long	:Int
FLTP	F	RFC_FLOAT	float ↔ double	:Real
CHAR+Länge	C(n)	RFC_CHAR	char[]	:String
DEC	P	RFC_PACKED	long	:packed.T
NUMC+Länge	N(n)	RFC_NUM	char*	:numeric.T
BINARY+Länge	X(n)	RFC_HEX	void* / tsp_OID	:binary.T
DATE	D(8)	RFC_DATE	int / char*	:sapDate.T
TIME	T(8)	RFC_TIME	int / char*	:sapTime.T

Tabelle 5.1: Abbildung der ABAP/4-Basistypen nach Tl

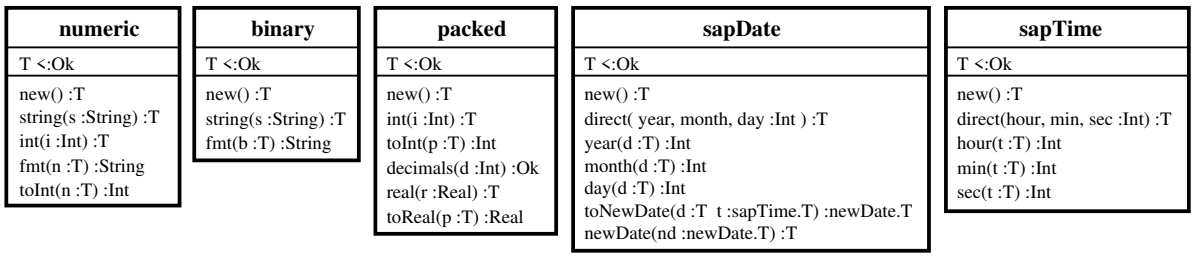


Abbildung 5.3: Strukturen der nichttrivalen ABAP/4-Basistypeinbettungen

oder sogar Domänen und Datenelemente (vgl. Abschnitt 3.2.2 Seite 42) erwartet, bietet sich deren Abbildung in die Sprache Tl an.

Tabelle 5.1 gibt einen Überblick über diese Typabbildung. Sie enthält im wesentlichen die Korrespondenz zwischen ABAP/4-Typ, C-Typ und dem entsprechenden Tl-Typ, als Repräsentationen in den Gateway-Schichten eins bis drei. Die ABAP/4-Typen `C`, `I` und `F` lassen sich dabei direkt nach Tl abbilden. Sie werden als direkte Parameter der C-Call-Schnittstelle an die Zwischenschicht zwei weitergereicht bzw. von dort nach Tycoon-Tl importiert.

Für den String-Typen `C(n)` ist die Umwandlung von einer Längenterminierung in ABAP/4 über die Programmiersprache C, die null-terminierte Strings besitzt, auf die längenterminierte Darstellung der Tl-Strings zu realisieren. Dazu wird der entsprechende Aktualwert direkt über Aufrufe der TSP-Schnittstelle in ein Tycoon-String-Objekt gespeichert und lediglich der damit erzeugte Objektidentifikator in den Tl-Kontext übergeben.

Alle weiteren ABAP/4-Typen besitzen eine spezielle Semantik, die von keinem der Tl-Basistypen gewährleistet wird, so daß sie auf die in Abbildung 5.3 aufgelisteten speziellen abstrakten Typen abzubilden sind. Ihre Übergabe an der Zwischenschicht zwei geschieht, wie in Tabelle 5.1 dargestellt, ähnlich dem Mechanismus für Strings nicht direkt über Parameter der C-Calls `bind()`-Funktion, sondern durch einen direkten Zugriff auf den Tycoon-Objektspeicher. Dies Vorgehen entspricht in der Klassifikation aus Abschnitt 2.2.2 Seite 19 dem Objektspeicherzugriff.

Behandlung von Tabellenparametern

Das in Abbildung 4.6 Seite 70 aufgeführte Modell für Funktionen der RFC-API unterliegt der gravierenden Einschränkung, nur homogene Typen `C` oder `X` als Typ für Zeilen der Tabellenparameter zuzulassen. Insbesondere sind keine Strukturen zugelassen. Dies würde jedoch einen Verlust jeglicher Schlüsselsemantik bedeuten und damit die Verwendung der Tabellenparameter auf einfache Fälle der Übergabe mehrfacher Werte der entsprechenden atomaren Typen beschränken.

Um dieses Problem zu umgehen, wurde eine Analyse der Kommunikationsbereiche für die Verwendung der übrigen Typen vorgenommen. Sie ergab, daß die Übertragung der Tabellenzeilen nicht

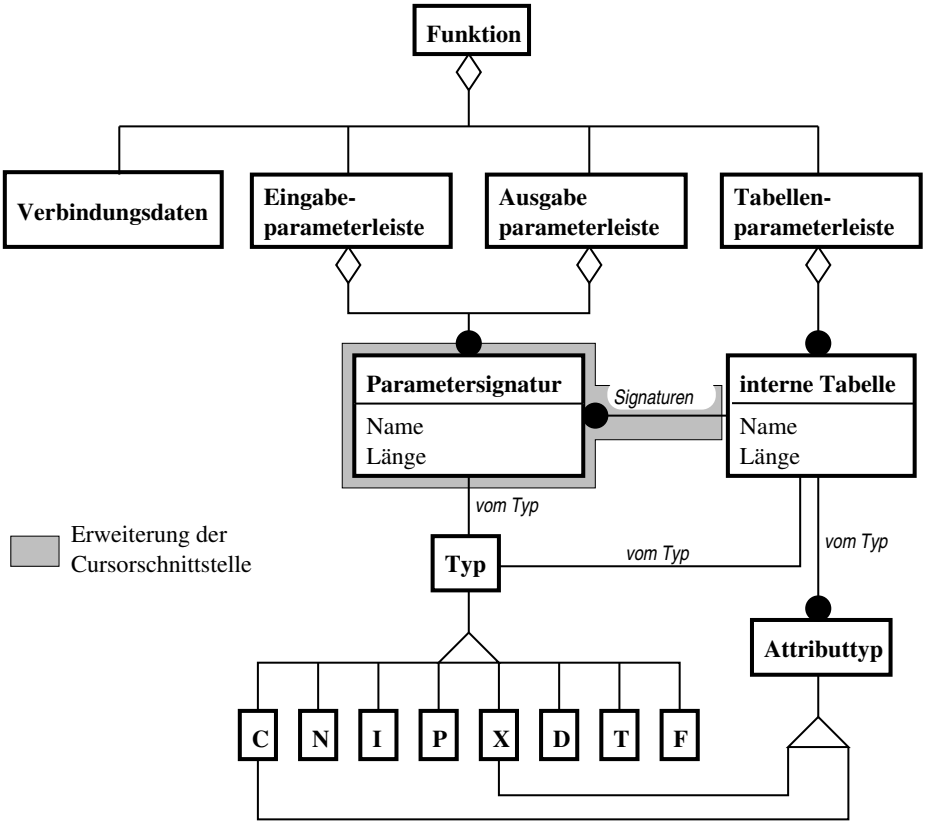


Abbildung 5.4: Erweitertes Objektmodell für externe RFC-Funktionen

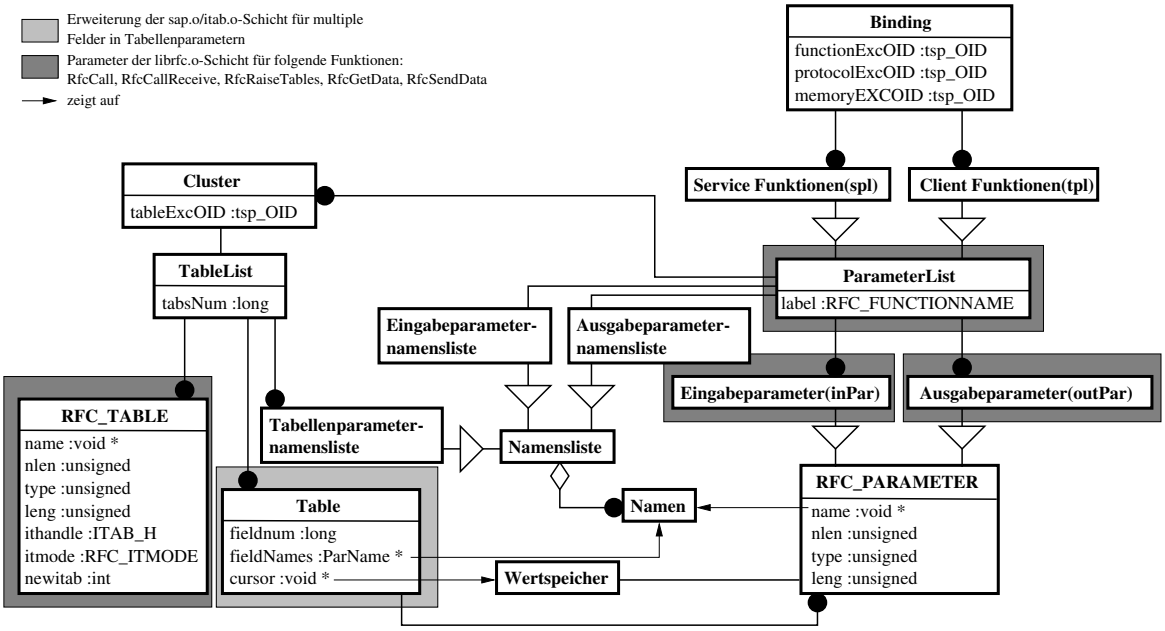


Abbildung 5.5: Parameter- und Tabellenstruktur für RFC-Aufrufe

an einzelnen Attributen orientiert ist, sondern lediglich eine Kopie der Speicherdarstellung angefertigt wird. Daher liegt nach der Übertragung die volle Information einer Tabellenzeile vor. Unter Kenntnis der internen Satzspeicherung können deshalb sukzessive alle Attributwerte ausgelesen werden. Als Nachteil ergibt sich dabei der Verlust der Datenunabhängigkeit, der aber dennoch als marginal anzusehen ist, weil alle bei einer Portierung notwendigen Änderungen bzgl. dieser Maschinenabhängigkeit innerhalb der ohnehin anzupassenden Zwischenschicht zwei liegen.

Der Algorithmus zum Auslesen der internen Sätze stützt sich auf die Annahme, daß alle Feldinhalte in der Definitionsreihenfolge abgelegt werden. Zusätzlich beginnen Werte der Typen Integer(ABAP/4-Typ `I`) und Float(ABAP/4-Typ `F`) auf an Wortgrenzen ausgerichteten Adressen.

Strukturen der Zwischenschicht

Die dynamische Bindung der Funktionsbausteine zur Laufzeit erfordert generische Prozeduren zur Erzeugung der Parameter- und Verwaltungsstrukturen der Zwischenschicht (vgl. Abschnitt 4.2.1). Diese müssen so ausgelegt sein, daß sie die Objektbeziehungen respektieren, die Persistenz und Mobilität des Tycoon-Systems unterstützen und einen möglicht geringen Einfluß auf das Laufzeitverhalten haben.

Abbildung 5.5 gibt eine Übersicht über alle bei der dynamischen Bindung relevanten Datenstrukturen mit ihren Interdependenzen. Eine konkrete Schnittstelle besteht aus den Bindeinformationen für mehrere Service- bzw. Client-Funktionen und den Objekt-Identifikatoren(OIDs) der Ausnahmen (vgl. Abschnitte 2.2.2,5.2.4), die durch die Anwendung, die RFC-Bibliothek bzw. die Speicherallokation ausgelöst werden. Diese Informationen sind in der `Binding`-Struktur zusammengefaßt.

Jede Funktion, unabhängig davon ob sie in der Client- oder Server-Rolle existiert, aggregiert selbst wieder Eingabe-, Ausnahme- und Tabellenparameter in der `ParameterList`-Struktur. Ein/Ausgabeparameter werden durch Arrays der `RFC_PARAMETER`-Strukturen repräsentiert, die neben ihrer Signatur Name, Typ und Byte-Länge auch eine Referenz auf den Speicher ihres Wertes enthalten. Die Namen der Parameter einer Parameterliste werden aus Performanzgründen selbst wieder listenartig in zusammengehörig alloziierten Speicherbereichen gehalten.

Die Tabellen-Parameter einer Funktion werden als `Cluster` bezeichnet. Das ist eine Struktur die neben dem OID für Ausnahmen der Tabellen-API eine `TableList`-Struktur enthält. Dort sind die

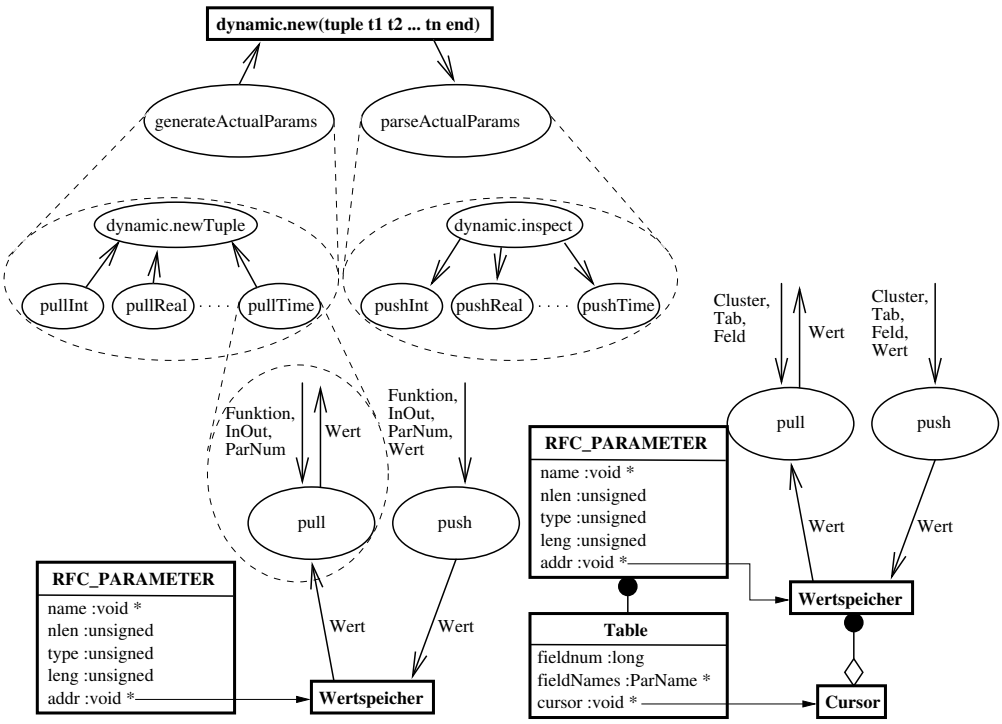


Abbildung 5.6: Polymorphes Marshalling der Aktualparameter

Anzahl der Tabellen, ein Array der `RFC_TABLE`-Parameterstrukturen, ein Array von `Table`-Strukturen und, wie bei den gewöhnlichen Parametern, eine Liste der Tabellenparameternamen abgelegt.

Die `RFC_TABLE`-Parameterstrukturen sind die eigentlichen Repräsentanten der Tabellenparameter, weil sie von entsprechenden Funktionen[2] der RFC-Bibliothek verarbeitet werden. Wie zuvor bereits erläutert, erlauben diese jedoch nicht heterogen attributierte Sätze beliebigen Typs. Diese zusätzlich erforderliche Information erhält man durch die `Table`-Struktur, die zu jeder Tabelle einen *Cursor* und eine Liste der Attribute enthält. Diese werden einfach als `RFC_Parameter`-Strukturen angelegt, deren Wertspeicherreferenz dann auf den entsprechenden Bereich des Cursors zeigt. So kann eine konkrete Tabellenzeile über die Kenntnis der Byte-Länge direkt in den Cursor-Bereich kopiert werden, dort durch die Information über die interne Satzdarstellung gelesen bzw. geändert und hiernach wieder in die Tabelle zurückgeschrieben werden.

Damit sind die Parameterstrukturen der Zwischenschicht in ihrem Aufbau und ihrer Bedeutung erklärt. Im folgenden wird der Algorithmus zur Transformation von Werten aus der Tycoon-Tl-Repräsentation auf die Wertzellen der Parameterstrukturen erläutert.

Polymorphes Parameter-marshalling bzw. -unmarshalling

Durch die fehlende gemeinsame statische Spezifikation der Funktionsbausteinschnittstellen und dem Tl-Programm einerseits und der daraus resultierenden dynamischen Bindung andererseits wird eine Typüberprüfung zur Übersetzungszeit ausgeschlossen. Daher müssen die Aktualparameter eines Funktionsbausteinaufrufs polymorph gemarshalled werden, was ihre Laufzeittyprepräsentation voraussetzt (→[Mat96a] S.26). Konkret werden Aktualparameterleisten als automorphe Werte des statisch prüfbaren Typs `dynamic.T` der Tl-Aggregation Tupel, wie in `dynamic.new(tuple t1 t2 ... tn end)`, angelegt (siehe Abbildung 5.6).

[2] `RfcCall()`, `RfcReceive()`, `RfcCallRecv()`, `RfcRaiseTables()`, ...

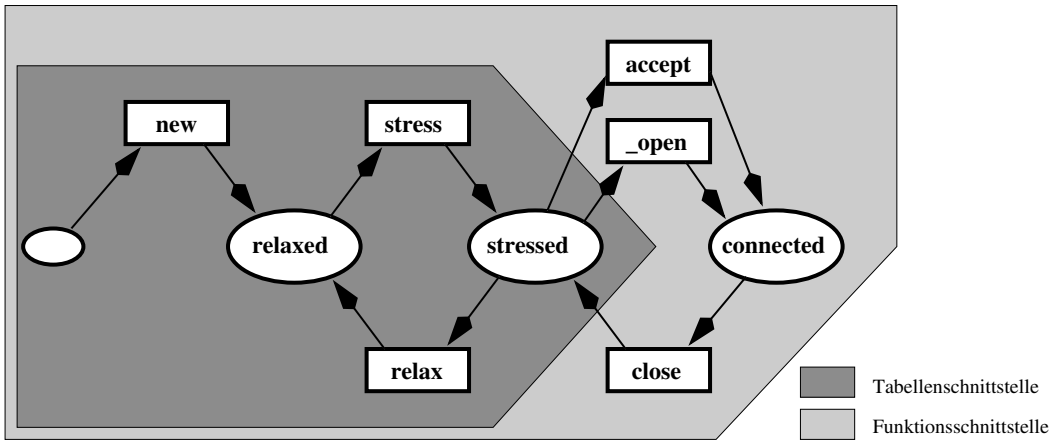


Abbildung 5.7: Unterstützung der Pseudo-Persistenz bzw. -Mobilität

Beim polymorphen Marshalling übernimmt die Funktion `parseActualParams()` ein Exemplar der Aktualparameterleiste und zerlegt diese entsprechend der im automorphen Wert gekapselten Typinformation in ihre Einzelparameterwerte, die mit der vorgegebenen Signaturspezifikation der Bindung typüberprüft werden. Im Erfolgsfall wird für jeden Einzelparameterwert eine typabhängige `push()`-Funktion aufgerufen, die ihrerseits an eine C-Funktion gebunden ist und mit der Funktionsnummer, einer In/Out-Information[3], einer Parameterplatznummer parametrisiert den Wert in die richtige Wertezelle marshalled.

Die inversen `pull()`-Funktionen, die ebenfalls an C-Funktion gebunden sind und mit der Funktionsnummer, einer In/Out-Information, einer Parameterplatznummer parametrisiert werden, liefern den revers gemarshallten Wert der Wertezelle des adressierten Parameters zurück. Aus den Werten aller Parameterpositionen wird dann mit `dynamic.newTuple()` der automorphe Tupelwert einer Ergebnisparameterleiste erzeugt, der von der Funktion `generateActualParams()` zurückgegeben wird.

Für Tabellenparameter wird das gleiche Prinzip der `push()`- bzw. `pull()`-Funktionen angewandt. Wegen der etwas differierenden Speicherrepräsentation der Tabellen (vgl. Seite 87), besteht die Wertzellenadressierung jedoch aus Angaben über das Cluster, die Tabellennummer und die Feldnummer.

5.2.2 Unterstützung von Persistenz und Mobilität

Eine wesentliche Anforderung der Tycoon-Umgebung ist die Unterstützung der orthogonalen Persistenz und Mobilität aller Objekte, insbesondere also auch des SAP-Gateways und ihrer Funktionsbausteinbindungen. Diese basieren auf der Ansteuerung der externen RFC-Bibliothek und erfordern die Verwaltung der im vorigen Abschnitt erläuterten flüchtigen Parameter- und Tabellenparameterstrukturen sowie weiterer Informationen zum Aufbau einer Verbindung. Dazu gehören Operationen zu deren Konstruktion und Destruktion. Die Standardmethode durch ein Loggen aller Konstruktoren und Destruktoren Pseudo-Persistenz bzw. -Mobilität herzustellen ist aus zwei Gründen an dieser Stelle nicht ausreichend:

1. Generell können lokale Tabellenparameter verarbeitet werden, ohne daß eine Verwendung selbiger in Funktionsbausteinen stattfindet. Umgekehrt setzt die Bindung von Funktionsbausteinen mit Tabellenparametern aber deren Instanziierung voraus.

2. Der Logging-Mechanismus ist nicht für das SAP-Gateway optimiert. Jede Operation würde differenziert für sich geloggt werden, was schon im Fall des C++-Gateways zu einem Leistungsnachteil von mehr als 25% geführt hat (→[Kor95]).

[3] Angabe darüber, ob es sich um eine Service- oder Client-Funktion handelt.

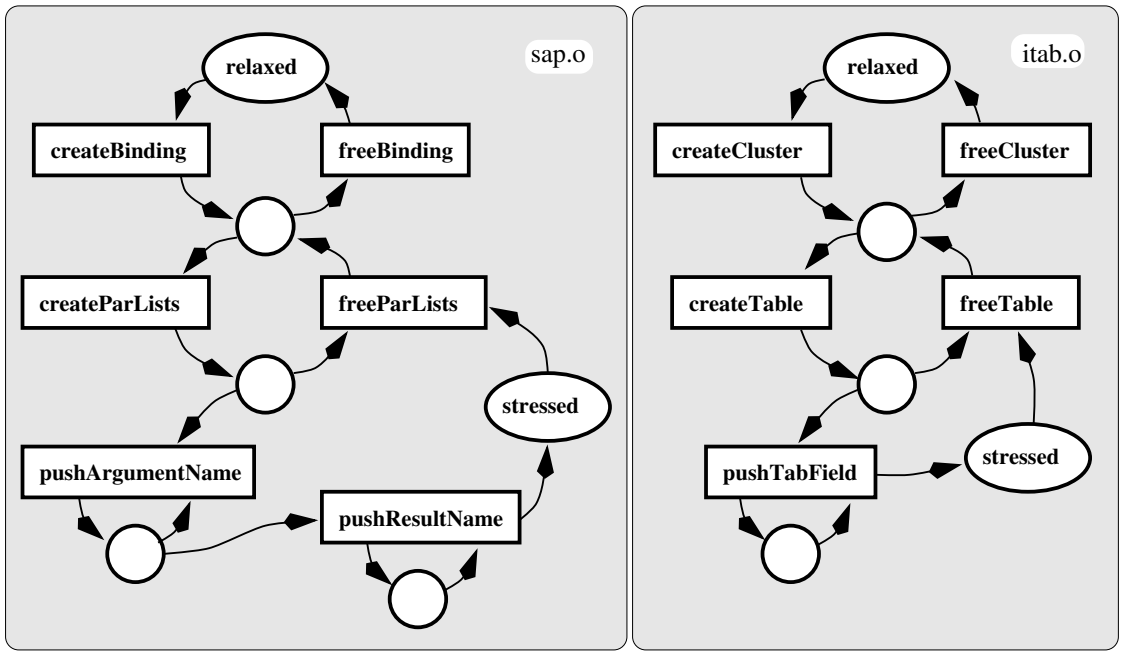


Abbildung 5.8: Dynamische Abläufe der Funktionen `stress()` und `relax()`

Daher wurden die drei Zustände **relaxed**, **stressed** und **connected** eingeführt (siehe Abbildung 5.7) und alle Struktur-Konstruktoren in der Operation `stress()` und alle Struktur-Destruktoren in der Operation `relax()` akkumuliert.

Jede Gateway-Instanz, bestehend aus einer SAP-Applikationsserver-Kennung und mehreren Client- bzw. Service-Funktionen, befindet sich nach ihrer initialen Erzeugung durch `new()` im Zustand **relaxed**. In diesem Zustand können neue Funktionsdefinitionen hinzugefügt oder bestehende gelöscht werden. Es sind aber keine flüchtigen Parameterstrukturen der Zwischenschicht angelegt, so daß volle Mobilität und Persistenz gewährleistet sind.

Durch Aufruf der Operation `stress()` wird in den Zustand **stressed** gewechselt und dabei entsprechend den dynamischen Modellen aus Abbildung 5.8 alle flüchtigen Parameterstrukturen der Zwischenschicht erzeugt. Damit ist die Verarbeitung der Tabellenparameter über das Tl-Modul `sapITAB.tm` und der Aufbau von aktiven oder passiven Verbindungen durch `_open()` oder `accept()` ermöglicht.

Wenn eine Verbindung zum SAP-Applikationsserver aufgebaut wird, wechselt die Gateway-Instanz in den Zustand **connected**. In diesem Zustand ist der Aufruf von Funktionsbausteinen bzw. die Diensterbringung durch Tl-Service-Funktionen aktiviert.

Durch `close()` wird eine bestehende Verbindung unterbrochen und `relax()` schaltet bei vollständiger Destruktion flüchtiger Objekte wieder in den Zustand **relaxed** zurück.

Für die Persistenz und Mobilität der Gateway-Instanz ist der Zustand **relaxed** entscheidend, der mit der Abwesenheit flüchtiger Parameterstrukturen äquivalent ist. Daher genügt es für das Operationslogging zur Pseudo-Persistenz bzw. -Mobilität die Operationen `stress()` und `relax()` zu registrieren (vgl. Abschnitt 2.1.2), was ein einmaliger Aufwand ist und somit nicht zu Performanzeinbußen führt:

```
persistent.create( stress   fun( v :Ok ) relax() )
```

5.2.3 Applikationschnittstellen

In den Abschnitten 4.2.3 und 4.2.4 ist die automatenbasierte Funktionsaufrufsteuerung der Gateway-Schicht eins respektive der RFC-Bibliothek dargestellt. Die Funktionen der Zwischenschicht zwei kapseln die Funktionen aus Schicht eins mit besonderer Berücksichtigung der Parameter, die seiteneffektbasiert aus der Strukturhierarchie unter der Binding-Struktur (vgl. Abbildung 5.5) referenziert werden. Deshalb besitzen alle Funktionen der Zwischenschicht zwei neben einer Referenz auf die Binding-Struktur nur atomar strukturierte Parameter, die einfach durch C-Calls der Sprache Tl in Gateway-Schicht drei abgebildet werden können (vgl. Abschnitt 2.2.2).

Im folgenden werden die abstrahierten Programmierschnittstellen der Gateway-Schicht drei erklärt. Dazu gehören neben den Operationen `stress()` und `relax()` des vorigen Abschnitts 5.2.2 der Tabellen-Parameter kapselnde abstrakte Datentyp `sapITAB.T`, sowie Funktionen zur SAP-Applikationsserver-identifikation, zur Deklaration von Client- und Service-Funktionen, zum Aufruf von Funktionsbausteinen und zur Inspektion der Gateway-Instanz in dem Modul `sapRFC.tm`.

Formalparameter

Die dynamische Bindung der Funktionsbausteine erfordert die Laufzeitrepräsentation der formalen Argument- und Ergebnisparameter sowie der Spezifikationen interner Tabellen-Parameter. Damit können zum Aufrufzeitpunkt die automorphen Aktualparameter typgeprüft werden, bevor sie polymorph gemarshalled werden (vgl. Abschnitt 5.2.1 S.88).

Die Signatur eines Formalparameters besteht aus seinem Namen, einer Typspezifikation und einer typabhängigen Byte-Länge, so daß folgendes drei-Tupel aus `SapCommon.it` eine passende Typbeschreibung liefert:

Let Param = **Tuple** name :**String** type :SapCommon.Type leng :**Int end**

`name` gibt den Parameternamen an. `type`, selbst vom Typ `SapCommon.Type`, legt dessen Typ fest. Hier stehen in Anlehnung an die zulässigen neun Funktionsbausteinparametertypen (vgl. Abschnitt 5.2.1 S.84) folgende Konstanten, gekapselt in `sapCommon.tm`, zur Auswahl:

sap_char, sap_string, sap_real, sap_int, sap_numeric,
sap_packed, sap_binary, sap_date, sap_time :Type

Die Länge `leng` wird für die Typen `sap_char, sap_real, sap_int, sap_date, sap_time` automatisch festgelegt und kann somit auf null gesetzt werden. Bei dem Typen `sap_string` gibt er die Länge vor. Die auf abstrakte Tl-Typen abgebildeten Typen `sap_numeric, sap_packed, sap_binary` müssen ebenfalls vorgegeben werden.

Daraus läßt sich die Darstellung von Parameterleisten für Argument und Ergebnis einer Funktion als

Let ParamArray = **Array**(Param)

ableiten. Entsprechend stellt sich die Spezifikation interner Tabellen-Parameter wie folgt dar:

Let Field = sapCommon.Param
Let Table = Tuple name :**String** fields :**Array**(Field) **end**

Tabellen-Parameter

Die Schnittstelle für Tabellen-Parameter orientiert sich an dem Funktionsmodell aus Abbildung 5.4. Daher kapselt ein Wert vom Datentyp `sapITAB.T` als Cluster (vgl. Abschnitt 5.2.1) mehrere Tabel-

len. Cluster sind Werte erster Klasse und werden bei der Deklaration von Funktionen in `sapRFC.tm` angegeben.

Ein initial leeres Cluster wird mit

```
new() :T
```

erzeugt. Entsprechend der Zustandssemantik aus Abschnitt 5.2.2 befindet es sich zunächst im Zustand **relaxed**. Es können Tabellendeklarationen mit

```
register( cluster :T  table :Table ) :TblContext
```

hinzugefügt oder gelöscht werden:

```
delete( cluster :T  name :String ) :Ok
```

Jedes Einfügen einer neuen Tabelle mittels `register()` liefert einen Wert der Tabellenoperationsschnittstelle zurück:

```
Let TblContext =
  Tuple
    appLine( row :dynamic.T ) :Ok
        (* Anhaengen einer Zeile an das Tabellenende *)
    insLine( line :Int  row :dynamic.T ) :Ok
        (* Einfuegen einer Zeile vor Position line *)
    delLine( line :Int ) :Ok
        (* Loeschen von Zeile line *)
    getLine( line :Int ) :dynamic.T
        (* Liefert die Zeile an Position line *)                    10
    putLine( line :Int  row :dynamic.T ) :Ok
        (* Aendern der Zeile an Position line *)
    fill() :Int
        (* Liefert Anzahl der Zeilen in der Tabelle *)
    free() :Ok
        (* Loeschen aller Zeilen *)
    elements() :iter.T(dynamic.T)
        (* Generiert eine Iteration aus der Tabelle *)
    create( it :iter.T(dynamic.T) ) :Ok
        (* Anhaengen einer Iteration an die Tabelle *)              20
  end
```

Ihre Verwendung setzt allerdings den Übergang des Clusters in den Zustand **stressed** voraus. Es gibt Funktionen zum Einfügen, Ändern, Löschen und Lesen von Tabellenzeilen. Außerdem kann eine Tabelle über `elements()` auf eine Tycoon-Tl-Iteration abgebildet werden. Durch `create()` hingegen wird eine Tl-Iteration an eine Tabelle angefügt.

Weitere administrative Operationen umfassen die Zustandssteuerung:

```
stress( cluster :T ) :word.T
relax( cluster :T ) :Ok
handle( cluster :T ) :word.T
```

`stress()` und `relax()` besitzen die in Abschnitt 5.2.2 beschriebene Funktionalität. Ergänzend wird für instanziierte Cluster ein Handle erzeugt, das als Referenz auf die angelegten flüchtigen Strukturen dient und an Funktionen des Moduls `sapRFC.tm` weitergereicht wird. Diese Handle kann bei

instanziierter Schnittstelle durch die Funktion `handle()` auch explizit abgefragt werden.

Erzeugen von Gateway-Instanzen und Funktionsregistratur

Neue Gateway-Instanzen werden durch

```
new( client, user, password :String
     language :Language  destination, hostname :String
     sysnr :Int  gateway_host, gateway_service :String ) :T
```

erzeugt. Die Parameter dienen unter Verwendung der Logon-Variante `RFC_MODE_R3ONLY` zur Identifikation und Adressierung des Servers (vgl. Abschnitt 4.2.3). Während eine Gateway-Instanz mit ihren Deklarationen an einen bestimmten SAP-Applikationsservice gebunden ist, die Adressierungsparameter also fest sind, können die Parameter zur Benutzeridentifikation nachträglich neu gesetzt werden:

```
setClient( sap :T  client :String ) :Ok
setUser( sap :T  user :String ) :Ok
setPassword( sap :T  password :String ) :Ok
setLanguage( sap :T  language :Language ) :Ok
```

Der Initialzustand eines Gateways nach seiner Erzeugung ist **relaxed**. In diesem Zustand können neue Client- bzw. Service-Funktionen deklariert und die vorliegende Instanz inspiziert werden. Im folgenden werden die zugehörigen Operationen, sowie der Aufruf von Funktionen im aktiven und passiven Fall vorgestellt.

Aktiver Funktionsaufruf

Für den aktiven Funktionsaufruf müssen alle Client-Funktionen entsprechend der Spezifikation in der Funktionsbausteinschnittstelle (vgl. Abschnitt 4.1.1) beim Gateway registriert werden. Dabei gelten die durch das Objektmodell für externe RFC-Funktionen gegebenen Einschränkungen (vgl. Abschnitte 4.2.1, 5.2.1).

Neue Client-Funktionen werden mit

```
registerClient( sap :T  label :String  arg, rslt :ParamArray
                cluster :sapITAB.T ) :Ok
```

in der Gateway-Instanz `sap` unter dem Namen `label` mit den Argumentparametern `arg`, den Ergebnisparametern `rslt` und den Tabellenparametern `cluster` deklariert. Unabhängig vom Zustand kann das Gateway jederzeit mit

```
clntElements( sap :T ) :iter.T(FunDesc)
```

inspiziert werden, um eine Iteration über die deklarierten Funktionen zu erhalten:

```
FunDesc <:Tuple
            label :String
            arg, rslt :ParamArray
            cluster :sapITAB.T
          end
```

Nach einem Wechsel des Gateways in den Zustand **stressed**, der transitiv für abhängige Tabellen-Cluster übernommen wird, ist nunmehr ausschließlich der Zugriff auf die Tabellen oder ein weiterer

Zustandswechsel erlaubt. Nach dem Wechsel des Gateways in den Zustand **connected** durch Öffnen einer Verbindung zum spezifizierten SAP-Applikationsserver mit `_open()` können Funktionen asynchron oder synchron aufgerufen werden.

Zum synchronen Aufruf muß ein Funktionswert der Signatur

```
SyncClntFun <:Fun( :dynamic.T ) :dynamic.T
```

mit synchroner Semantik über den Namen der Service-Funktion erzeugt werden:

```
getSyncClientFun( sap :T  label :String ) :SyncClntFun
```

Ein synchroner Funktionsaufruf blockiert nun solange, bis der entsprechende Funktionsbaustein abgearbeitet ist und das Funktionsergebnis vorliegt, welches dann als Wert zurückgeliefert wird.

Der asynchrone Aufruf funktioniert ähnlich. Zuerst muß ein asynchroner Funktionswert der Signatur

```
ASyncClntFun <:Fun( :dynamic.T ) :Result
```

durch Aufruf von

```
getASyncClientFun( sap :T  label :String ) :ASyncClntFun
```

erzeugt werden. Dann wird die Applikation mit entsprechenden Aktualarametern durchgeführt, wobei die Funktion nicht blockiert, sondern ein Tupel des Typs

```
Result <:Tuple
          snoop :Fun():Bool
          receive :Fun():dynamic.T
        end
```

zurückliefert. Über die Funktion `snoop()` kann daraufhin geprüft werden, ob bereits ein Funktionsergebnis vorliegt und dieses dann mittels `receive()` geholt werden. Andernfalls, wenn `snoop()` ein `false` zurückgibt, blockiert `receive()` wie im synchronen Aufruf bis zum Eintreffen des Ergebnisses.

Passiver Funktionsaufruf

Tycoon-Tl kann als Diensterbringer für SAP R/3 Funktionsaufrufe dienen. Dazu müssen Service-Funktionen der Signatur

```
SrvcFun <:Fun( :T :dynamic.T ) :dynamic.T
```

angelegt werden. Diese können dann mit

```
registerService( sap :T  func :SrvcFun
                 label :String  arg, rslt :ParamArray
                 cluster :sapITAB.T  docu :String ) :Ok
```

bei der Gateway-Instanz `sap` unter dem Namen `label` mit der Kurzdokumentation `docu` (vgl. Abschnitt 4.2.4) registriert werden oder durch

```
remove( sap :T  label :String ) :Ok
```

wieder entfernt werden. In jedem Gateway-Zustand kann eine Iteration über alle registrierten Service-Funktionen

```
SrvcFunDesc <:Tuple
                fd :FunDesc
                sf :SrvcFun
                docu :String
              end
```

durch

```
srvcElements( sap :T ) :iter.T(SrvcFunDesc)
```

erzeugt werden.

Nach einem Wechsel in den Zustand **stressed** kann das Gateway durch den Aufruf von `accept()` in die Diensterbringerrolle versetzt werden. Es wird dann automatisch beim Kommunikations-Gateway registriert und eingehende Rufe werden auf die beim Gateway registrierten Service-Funktionen verteilt. Historisch bedingt beendet das Gateway die Verbindung zum Kommunikations-Gateway nach einem Funktionsaufruf, so daß zur Verarbeitung mehrfacher Aufrufe eine Schleife zu programmieren ist:

```
loop
  accept( sap )
  if abbruch then exit end
end
```

Die Unterscheidung zwischen asynchronem und synchronem Aufruf wird im ABAP/4-Client durch die richtige Aufrufform getroffen. Durch

```
CALL FUNCTION label STARTING NEW TASK taskid DESTINATION logdest
```

wird ein asynchroner Aufruf und durch

```
CALL FUNCTION label STARTING DESTINATION logdest
```

ein synchroner Aufruf durchgeführt (vgl. Abschnitt 4.1.2 und Tabelle 4.2).

5.2.4 Behandlung von Fehlersituationen durch Ausnahmen

Die Verwendung des Tycoon-SAP R/3-Gateways unterliegt gewissen Unsicherheiten, die sich daraus ergeben, daß

- in dem aufgerufenen Funktionsbaustein zustandsorientierte Fehler erkannt werden und daraufhin eine Ausnahme ausgelöst wird.
- es zu Problemen im System oder Netzwerk kommt, die von der RFC-Bibliothek signalisiert werden.
- bei dem Übergang von Zustand **relaxed** zum Zustand **stressed** des Gateways in Zwischenschicht zwei benötigter Speicher nicht vorhanden ist.
- Gateway-Funktionen im falschen Zustand aufgerufen werden.
- bei der Deklaration von Funktionen beim Gateway fehlerhafte Formalparameter angegeben werden oder Aktualparameter nicht mit den spezifizierten Formalparametern übereinstimmen.
- passiv aufgerufene Funktionen nicht definiert sind.

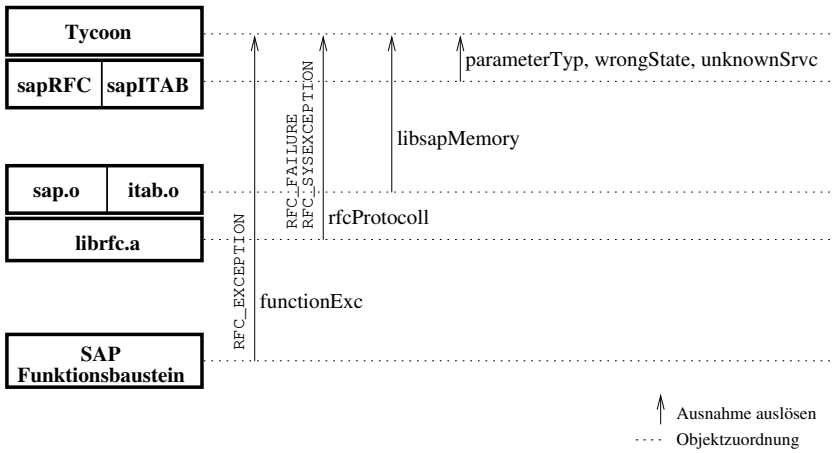


Abbildung 5.9: Hierarchie möglicher Ausnahmen

Der Mechanismus zur Behandlung solch unvorhergesehener Fälle sind Ausnahmen als Abstraktion primitiver Funktionen, die Fehlerereignisse über spezielle Rückgabewerte melden. Die oben aufgeführten Fehlersituationen werden durch folgende Ausnahmen repräsentiert:

```
functionExc    :Exception description :String end
rfcProtocol    :Exception operation, key, status, message, intstat :String end
libsapMemory   :Exception description :String  num :Int end
wrongState     :Exception inFunction :String  state :State end
parameterTyp   :Exception description :String end
unknownSrvc    :Exception name :String end
```

Abbildung 5.9 zeigt die Zuordnung der Ausnahmen zu ihrem Ursprung, der jeweiligen Gateway-Schicht bzw. dem SAP-Applikationsserver. Die letzteren drei Ausnahmen werden direkt auf der Tl-Sprachebene ausgelöst. Dagegen sind die ersteren drei das Ergebnis von bestimmten Funktionen der RFC-Bibliothek aus Gateway-Schicht eins oder von Speicherallokationsfunktionen der Zwischenschicht zwei. Die Rückgabewerte der RFC-Bibliotheksfunktionen, die außer den Fehlern mittels `RFC_CALL` auch noch einen Rückruf anzeigen, werden durch die Funktion `rc_eval()` der Zwischenschicht zwei gefiltert (siehe Abbildung 5.10).

Im Fall eines Rückrufs wird die Funktion `RfcDispatch()` der RFC-Bibliothek aufgerufen, die den Kontrollfluß an die adressierte Service-Funktion weiterleitet (vgl. Abschnitt 4.2.4).

Anwendungsverursachte Ausnahmen, die im aufgerufenen Funktionsbaustein durch `RAISE` ausgelöst wurden, werden durch den Mechanismus aus 2.2.2 direkt in die Tl-Ausnahme `functionExc` der Gateway-Schicht drei übersetzt.

Für die verbleibenden Fälle der System- oder Protokoll-Fehler greift die weitere Filterfunktion `rfc_error()`, die über `RfcLastError()` zusätzliche Information zur Fehlerursache ermittelt und diese dann ebenfalls als Tl-Ausnahme `rfcProtocol` auslöst.

5.3 Erweiterung um statische Typprüfung

Bereits in der SQL-Schnittstelle aus [Kor93] (vgl. Abschnitt 2.3.1) ist gezeigt worden, daß sich die dynamische Bindung externer Objekte durchaus statisch typisieren läßt, sofern zur Übersetzungszeit eine Typspezifikation der Objekte verfügbar ist. Dies wird dort durch die Verwendung eines Typoperators, parametrischem Polymorphismus und einer Objektprobe des entsprechenden Typs erreicht.

Für die Tycoon-SAP-Gateway-Schicht vier wird reflektiver Polymorphismus eingesetzt, der die

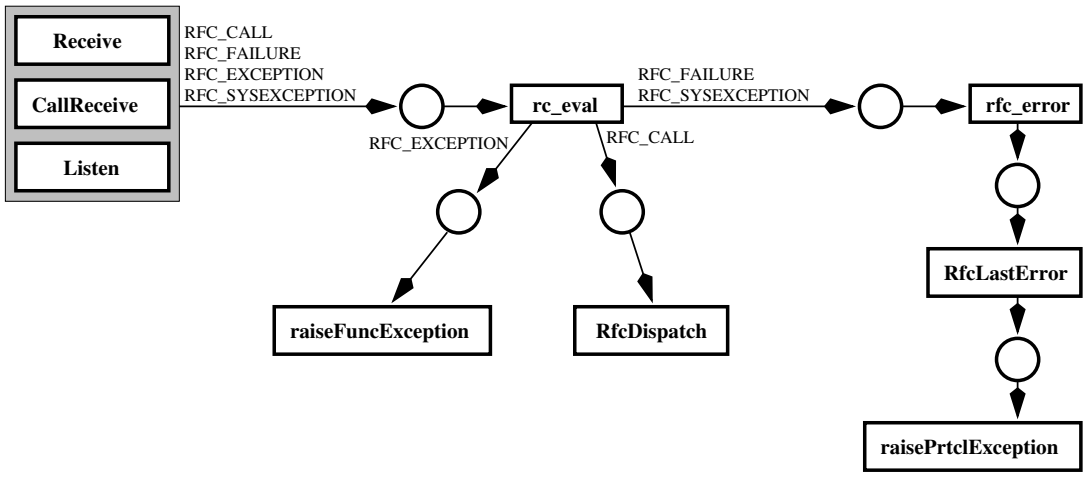


Abbildung 5.10: Ausnahmefilter der Schicht zwei

Typunsicherheiten (vgl. Abschnitt 2.3.1) der SQL-Schnittstelle vermeidet, weil er neben dem korrekten statischen Typ auch eine vollständige dynamische Typspezifikation erhält, die sonst notwendige und typunsichere Hilfskonstruktionen zur Typüberprüfung über das Objektspeicherformat der o.g. Objektprobe vermeidet.

Die Schnittstellenmodule `sapITAB.tm` und `sapRFC.tm` der Gateway-Schicht drei orientieren sich an dem Funktionsmodell und dem dynamischen Aufrufprotokoll der RFC-Bibliothek. Formalparameter werden durch unhandliche drei-Tupel und Aktualparameter durch komplizierte automorphe Werte angegeben. Diese Nachteile werden in der Gateway-Schicht vier beseitigt, so daß gebundene SAP R/3-Funktionsbausteine bzw. Tycoon-Tl-Service-Funktionen weitgehend statisch typisierten Tl-Funktionen entsprechen. Damit wird die dynamische Bindung mit ihren automorphen Werten zu einem generischen Stub degeneriert, der vollständig abstrahiert wird.

5.3.1 Typ- und Signaturkonstruktoren

Die Grundlage der vereinfachten Typrepräsentation für formale Parameter oder Tabellenattribute ist ihre abstrakte Darstellung als

Attribute <:**Ok**

mit den zugehörigen Konstruktorfunktionen für jeden der möglichen neun Typen (vgl. Abschnitt 5.2.1):

int(label :**String**) :Attribute
char(label :**String**) :Attribute
real(label :**String**) :Attribute
string(label :**String** len :**Int**) :Attribute
numeric(label :**String** len :**Int**) :Attribute
packed(label :**String** len :**Int**) :Attribute
binary(label :**String** len :**Int**) :Attribute
date(label :**String**) :Attribute
time(label :**String**) :Attribute

Jede dieser Konstruktorfunktionen übernimmt als Parameter nur die notwendigen Charakteristika bestehend aus dem Namen `label` und eventuell einer Längenangabe `len` ihres Typs.

Für Eingabe-, Ausgabe- und Tabellenparameter sind die *Attribute* derart zu aggregieren, daß jeweils

ihre Signatur beschrieben wird. Insbesondere müssen also Konstruktor-Funktionen realisiert werden, die Tl-Funktionen mit verschiedenzahligen Attributen im Argument- bzw. Ergbnisbereich und Tabellen unterschiedlichen Grades erzeugen. Das jedoch steht im Widerspruch zu den Tl-Spracheigenschaften:

- Funktionsobjekte haben eine durch ihren Typ vorgegebene nicht variable Anzahl Argumentparameter.
- Funktionsobjekte haben exakt einen Ergebnisparameter.
- Tupel haben eine feste durch ihren Typ vorgegebene Anzahl von Einträgen.

Deshalb müssen Konstruktor-Funktionen abhängig von der Signaturlänge des Objekts angeboten werden. Außerdem müssen strukturierte Funktionsergebnisse abgebildet werden. Und die Spezifikation der Attribute im Argument- und Ergebnisbereich für Funktionen muß unkorreliert sein, damit sich die Anzahl der Konstruktorfunktionen nicht als Produkt, sondern als Summe der Signaturlängen ergibt.

5.3.2 Strukturtypen für Tabellen-Parameter

Grundlage der statisch typisierten Repräsentation Tabellen-Parameter ist der Typoperator

```
Table( T <:Ok ) <:Ok
```

der durch die Konstruktor-Funktionen der Art

```
table0( name :String ) :Table(Tuple end)
table2( Dyn F1, F2 <:Ok   name :String   f1, f2 :Attribute ) :Table(Tuple :F1 :F2 end)
```

erzeugt wird. Dabei gibt die Nummer hinter der `table`-Anweisung die Anzahl der Attribute der entsprechenden Tabelle an. Außerdem wird eine Tabelle durch ihren Namen `name` identifiziert und besitzt eine Signatur aus ihren Attributen `f1, f2, ..,fn :Attribute`. Es sind alle Konstruktor-Funktionen bis zur Signaturlänge sieben vordefiniert.

In Funktionsaufrufen des Gateways sind jedoch nicht einzelne Tabellen sondern ganze Cluster aus mehreren Tabellen als Parameter anzugeben, so daß unter Verwendung der oben genannten Tabellenkonstruktoren durch

```
newCluster2( Row1, Row2 <:Ok   tab1 :Table(Row1)   tab2 :Table(Row2) )
             :Cluster2( Row1 Row2 )
```

ein neues Tabellen-Cluster vom Typ

```
Cluster2( Row1, Row2 <:Ok ) <:Tuple
                                table1 :TabAccess(Row1)
                                table2 :TabAccess(Row2)
                              end
```

mit zwei Tabellen generiert wird. Über den Tupel-Selektor kann aus einem Cluster eine Tabelle ausgewählt und auf ihre Schnittstelle zugegriffen werden:

```
TabAccess( Row <:Ok ) <:Tuple
  appLine( row :Row ) :Ok
        (* Anhaengen einer Tabellenzeil *)
  insLine( line :Int   row :Row ) :Ok
        (* Einfuegen einer Tabellenzeile *)
  delLine( line :Int ) :Ok
```

```
        (* Loeschen einer Tabellenzeile *)
    getLine( line :Int ) :Row
        (* Wert einer Tabellenzeile bestimmen *)
    putLine( line :Int  row :Row ) :Ok
        (* Eine Tabellenzeil ueberschreiben *)
    fill() :Int
        (* Anzahl der Tabellenzeilen ermitteln *)
    free() :Ok
        (* Loeschen aller Tabellenzeilen *)
    elements() :iter.T(Row)
        (* Eine Iteration aus allen Tabellenzeilen generieren *)
    create( it :iter.T(Row) ) :Ok
        (* Alle Eintraege der Iteration an die Tabelle anhaengen *)
end
```

Der Zugriff auf diese Tabellen-Schnittstelle ist natürlich nur verfügbar, wenn sich die Gateway-Instanz im Zustand **stressed** befindet.

5.3.3 Abbildung von Funktionen variabler Parameterzahl in Tl

Komplizierter als das Design für die Tabellen-Parameter ist das einer polymorphen `bind`-Funktion, aus den bereits oben angedeuteten Gründen:

- Client- und Service-Funktionen verschiedener Parameterzahl müssen gebunden werden.
- Mehrfache Ergebnisparameter sind notwendig.

Letzteres Problem läßt sich einfach durch Tupel-wertige Funktionsergebnisse abbilden. Ersteres Problem in Kombination mit der Forderung nach unkorrelierten Ergebnis- und Argument-Konstruktoren erfordert vier Konstruktorfunktionsarten, unterschieden nach Argument- bzw. Ergebnissignaturen und Client- bzw. Service-Funktionen.

Für Client-Funktionen beim aktiven Funktionsaufruf ist der parametrisierte Typoperator

```
Result( T <:Ok ) <:Ok
```

als Abstraktion eines Funktionsergebnistypen das Bindeglied zwischen Ergebnis-Konstruktoren, wie

```
result2( Dyn R1, R2 <:Ok  r1, r2 :Attribute ) :Result(Tuple :R1 :R2 end)
```

und `bind`-Konstruktorfunktionen, wie

```
bindFun2( Dyn I1, I2 <:Ok  O <:Ok  sap :T  label :String
          i1, i2 :Attribute  o :Result(O)  tabCluster :cluster.T ) :Fun( p1 :I1  p2 :I2 ) :O
```

Beide sind entsprechend der Forderung orthogonal kombinierbar und bis zu der Signaturlänge sieben vordefiniert.

Für Service-Funktionen beim passiven Funktionsaufruf kapselt der semiabstrakte Typ

```
CallRslt <:Tuple
            rslt :sapRFC.ParamArray
         end
```

die Signatur eines Service-Funktionsergebnisses. Diese Typ wird durch Konstruktoren der Art

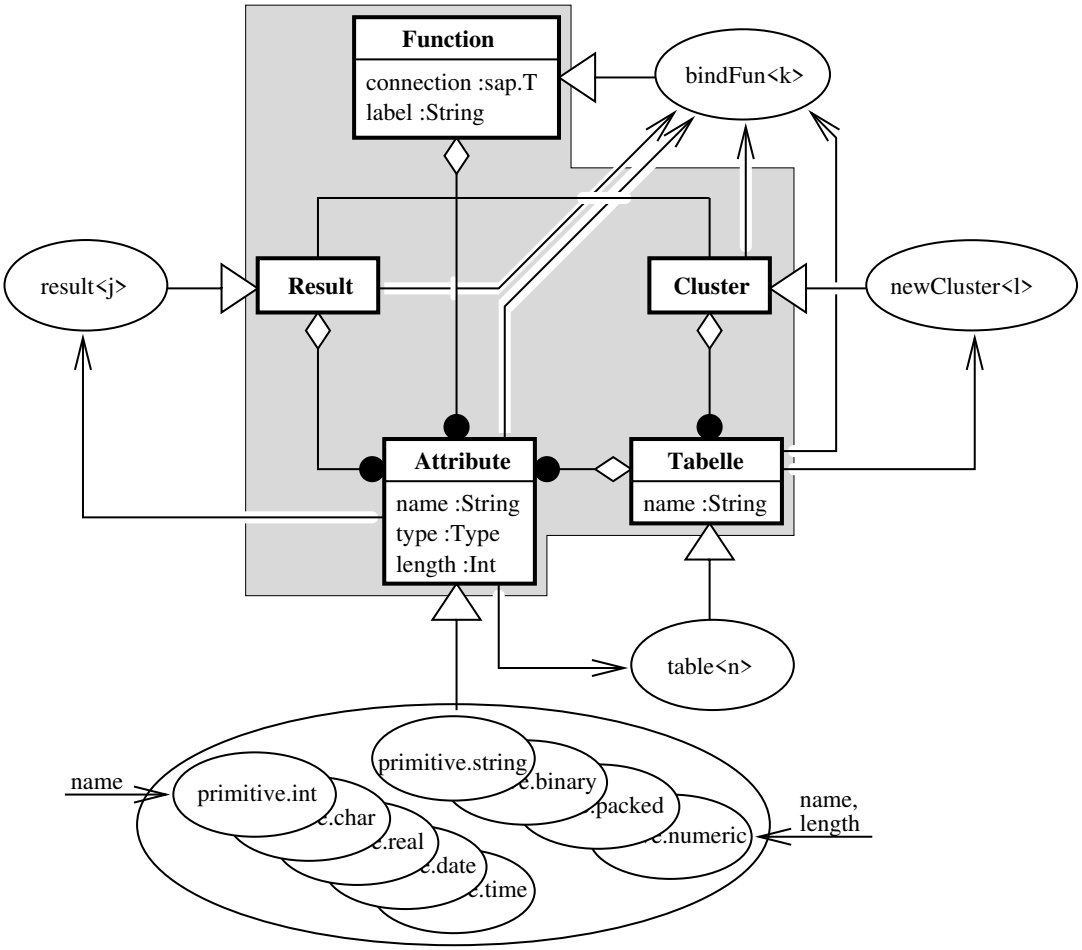


Abbildung 5.11: Abstraktes Objektmodell der Gateway-Schicht vier

```
callRslt2( r1, r2 :Attribute ) :CallRslt
```

erzeugt und in den Funktionen zur Service-Funktions-Registratur verwendet:

```
installFun2( Dyn I1, I2, O <:Ok sap :T  label :String
             i1, i2 :Attribute  cRslt :CallRslt  tabCluster :cluster.T
             function :Fun( :I1 :I2 ) :O ) :Ok
```

Wie für Client-Funktionen sind auch die Konstruktoren der Service-Funktionen orthogonal kombinierbar. Ebenfalls sind sie bis zur Signaturlänge sieben vordefiniert.

Abbildung 5.11 bietet einen zusammenfassenden Überblick auf die Objekte und Konstruktoren der Gateway-Schicht vier.

Kapitel 6

Zusammenfassung der Ergebnisse und Ausblick

Als Grundlage dieser Arbeit dienten die Systeme SAP R/3 und Tycoon-Tl. SAP R/3 präsentiert sich als Multi-Prozess-Middleware für betriebswirtschaftliche Anwendungen. Dabei wird von SAP ein hoher Integrationsgrad angestrebt. Alle betriebswirtschaftlichen Module sind vollständig in der 4GL-Sprache ABAP/4 unter Verwendung der semantischen Objekte SAP-Tabellen, ABAP/4-Programme und SAP-Transaktionen geschrieben. Die wichtigste Einschränkung dieses *build in*-Ansatz ist die mangelhafte Erweiterbarkeit gegenüber neuen Diensten, wie z.B. dem Internet, und neuen Technologien, wie polymorphen Datenmodellen und Code-Mobilität. Deshalb stellte sich Frage mit welchen Mitteln die Skalierbarkeit und Offenheit des SAP R/3 verbessert werden kann.

Im Vergleich bietet die Datenbankprogrammiersprache Tl ein hohen Grad an Generizität, Persistenz, Mobilität, Skalierbarkeit und Offenheit. Sie wird deshalb als Sprache zur Systemverschmelzung betrachtet. Durch Integration von Fremdsystemen in die Tycoon-Tl-Programmierwerkbank entsteht ein Framework verschiedenartiger Applikationen, deren Verwendung nur wenig mehr als den relativ schmalen Sprachkern von Tl vorraussetzt. Als Beispiele bereits integrierter Fremdsysteme wurden die SQL-Schnittstelle, Stp/OMT, NFS und INQUERY herangezogen. Diese unterscheiden sich neben ihrer Applikationssemantik in den verwendeten Bindungstechniken, der Typsicherheit, in der Unterstützung von Persistenz bzw. Mobilität und in der Aufrufrichtung, das heißt die Applikation tritt in der Client- oder Server-Rolle gegenüber dem Tycoon-System auf.

Es ergaben sich die beiden grundlegenden Bindungstechniken der Stub-Code-Generierung aus einer statischen Spezifikation der Schnittstelle des zu bindenden Objekts und der dynamischen Bindung aus einer Laufzeitrepräsentation der Schnittstellensignatur. Nachteilig für die dynamisch Bindung ist die erforderliche Generik der verwendeten Werte, die dementsprechend als automorphe Werte mit ihrer inhärenten Laufzeitineffizienz[1] zu repräsentieren sind. Vorteilig hingegen ist die größere Flexibilität der dynamischen Bindung, weil jegliche Spezifikation auch zur Laufzeit noch generiert werden kann.

Desweiteren wurden diese Implementationen daraufhin untersucht, wie die durch externe Bindungen über die C-Calls-Schnittstelle entstehenden Probleme bzgl. Persistenz und Mobilität gelöst wurden. Sie ergeben sich durch die Erzeugung flüchtiger Datenobjekte außerhalb der Kontrolle des Tycoon-Objektspeichers und führen beim Wiederanlauf nach einem Systemausfall oder bei einem Objekttransport zu hängenden Referenzen. Durch die Verwendung von Logging-Mechanismen für die Objektgenerierungs- bzw. Objektdegenerierungsfunktionen wird dieses Problem adäquat gelöst.

Für die Implementierung weiterer Gateway-Schnittstellen ergaben sich *Applikationstransparenz*, *Typsicherheit*, *Generizität* und *Datenunabhängigkeit* als wichtige Eigenschaften einer Systemintegration. Sie gewährleisten einen möglichst transparenten Zugriff auf die Applikationen unter Wahrung der Vorteile des Tycoon-Systems und der Allgemeinheit der Einbindung, die nicht durch die spezielle Art

[1] Typprüfungen erfolgen zur Laufzeit.

einer bestimmten Applikation dominiert werden sollte. Durch die Datenunabhängigkeit wird zusätzlich die Selbständigkeit der Informationen im Applikationssystem garantiert.

6.1 Strukturen und Schnittstellen des SAP R/3

Es erhob sich die Frage nach der Integration des SAP R/3 in die Tycoon-Tl-Entwicklungsumgebung. Eine Analyse des *Prozeßmodells*, der Entwicklungsumgebung, der semantischen Objekte des SAP R/3, der Programmiersprache ABAP/4 und der offenen Schnittstellen des SAP R/3 lieferte ein genaues Bild der Architektur. Die wichtigsten semantischen Objekte *SAP-Tabellen*, *ABAP/4-Programme* und *SAP-Transaktionen* repräsentieren die Design-Vorlagen aller SAP-Module. SAP-Tabellen abstrahieren Datenbanktabellen und werden durch Open-SQL-Befehle über die Datenbankschnittstelle inspiziert und modifiziert. ABAP/4-Programme werden vom ABAP/4-Prozessor interpretiert und SAP-Transaktionen sind Folgen von Dynpros, die vom Dynpro-Prozessor interpretiert werden und weitegehend dem Modell der Open Nested Transactions entspricht.

Unter vielen Schnittstellen, wie die Dateischnittstelle, CPI-C und RFC, BAPIs, EDI und ALE, erwies sich nur die Remote Function Call-Schnittstelle als hinreichend offen und verfügbar, um den Transparenzanforderungen gerecht zu werden. Insbesondere wurde gezeigt, daß über den RFC ein Zugriff auf die semantischen SAP R/3-Objekte möglich ist, als Open-SQL-Befehle durchgeführt, ABAP/4-Programme gestartet und SAP-Transaktionen gesteuert werden können.

Die Möglichkeiten des RFC umfassen den bidirektionalen entfernten Funktionsbausteinaufruf mit Eingabe-, Ausgabe- und Tabellenparametern. Dabei werden eine asynchrone, eine synchrone und eine transaktionale Aufrufsemantik unterstützt. Zur Programmierung für externe Systeme bietet SAP eine Programmierschnittstelle in Form der RFC-Bibliothek an. Es wurden *dynamische Ablaufmodelle* und ein *Objektmodell* für die Verwendung dieser Bibliothek erstellt. Sie besitzt einige Einschränkungen hinsichtlich der Parametersemantik und der zugelassenen Paramtertypen, die sich auf atomare Typen beschränken. Das wiederum hat den Verlust der Schlüsselsemantik für Tabellenparameter zur Folge.

6.2 Integration über ein vierschichtiges Gateway

Für die Integration des SAP R/3 über die Funktionsbibliothek respektive RFC in das Tycoon-Tl-System wurde eine *vierschichtige Gateway-Architektur* entworfen, die auf der RFC-Bibliothek als untersten Schicht basiert. Zur Bindung der Funktionen wurde die dynamische Laufzeitbindung gewählt, weil nicht a priori von dem Vorhandensein einer statischen Spezifikation der Funktionsbausteinschnittstellen ausgegangen und außerdem eine hohe Änderungsfrequenz für Funktionsbausteine vorausgesagt wurde.

Damit unterlag die Gateway-Konstruktion drei wesentlichen Aspekten:

- Das statische Funktionsmodell der RFC-Bibliothek mit seinen Parametern, Typen und Objekten mußte in Tl repräsentiert werden, was insbesondere eine Abbildung der ABAP/4-Typen auf Tl-Typen, Marshalling-Routinen für Aktualparameterwerte und Mechanismen für die Persistenz- bzw. Mobilitätsunterstützung erforderte.

- Die dynamischen Steueroperationen der RFC-Bibliothek mit ihren rückgabewertbasierten Zustands- und Ausnahmemeldungen mußten auf passende Tl-Abstraktionen, wie die Callback-Bindung oder den Ausnahme-Mechanismus, abgebildet werden.

- Die zur dynamischen Bindung erforderlichen automorphen Werte mit ihren umständlichen Inspektionsalgorithmen sollten durch eine generische, statisch typisierte Schnittstelle verborgen werden.

Die wurde in einem vierschichten-Modell realisiert. Die unterste Schicht eins wurde durch die Operationen, Funktionen und Objekte der RFC-Bibliothek vorgegeben. Schicht zwei ist eine Zwischen-

schicht, die der Abstraktion der RFC-Bibliotheksfunktionen dient und Parameterstrukturen verwaltet, sowie Ausnahmen und Rückrufe propagiert. Schicht drei bindet die Funktionen und Objekte der in C geschriebenen Schicht zwei an Tl-Funktionen und realisiert zustandsgesteuert Persistenz und Mobilität. Auf dieser Ebene sind unter Verwendung automorpher Werte bereits Client- und Service-Funktionsbindungen möglich. Zur Vereinfachung der Programmierung mit dem Gateway wurde in Schicht vier eine generische Stub-Schnittstelle geschaffen, die über Konstruktor-Funktionen statisch typisiert Tabellen und Client- bzw. Service-Funktionen bindet.

Damit ist der RFC, der die Grundlage verteilter Client/Server-Programmierung des SAP R/3 bildet, in das System Tycoon-Tl integriert worden. Aus Tl können nun Funktionsbausteine und aus SAP R/3 Tl-Funktionen aufgerufen werden. Außerdem wird auch der mehrfache Rückruf unterstützt. Desweiteren finden die erwähnten Eigenschaften für Gateway-Integrationen Anwendung:

Applikationstransparenz Funktionsbausteine sind vollständig abgebildet und deshalb als Anwendung auch transparent.

Typsicherheit Alle Aktualparameter werden dynamisch typgeprüft mit einer expliziten Unterscheidung zwischen neun möglichen Typen.

Generizität Das Gateway ist nicht auf die Anwendung bestimmter Funktionsbausteine vorkonzipiert, sonderen läßt beliebige Bindungen im Rahmen der RFC-Bibliothekseinschränkungen zu.

Datenunabhängigkeit SAP-Applikationen und ihr Datenmodell sind durch die Art der Schnittstelle nicht beeinflußt.

6.3 Ausblick und offene Probleme

Weitergehende Aufgaben der entwickelten Gateway-Integration liegen in der Verbesserung der technischen Eigenschaften des Gateways selbst, in der Abstraktion der Gateway-Funktionalität hin zu den semantisch reicheren Objekten des SAP R/3 und einem anderen Ansatz, der nicht den RFC als Interaktionsschnittstelle des SAP R/3 nutzt, sonderen die *Intelligent Terminal*-Schnittstelle, die algorithmengesteuert direkten Zugriff auf SAP-Transaktionen realisiert.

Weiterentwicklung der Gateway-Implementation

Für die Gateway-Implementation bieten sich Verbesserungsmöglichkeiten aufgrund der gemachten Erfahrungen und der durch SAP weiterentwickelten und inzwischen besser dokumentierten RFC-Bibliothek.

Die für externe RFC-Aufrufe zugelassenen Parameter sind zur Zeit atomar, obwohl die Übertragung der Strukturen seitens des SAP-Applikationsservers unterstützt[2] wird, wenn diese in der Funktionsbausteindefinition angegeben werden. Die Erkenntnisse über die interne Satzspeicherung bei Tabellenparametern können auf normale Parameter übertragen werden, so daß beliebige Strukturen im SAP-Sinn für alle Parameter zulässig sind.

Seit Version 3.0 unterstützt SAP R/3 den transaktionalen entfernten Funktionsbausteinaufruf mit einer für Datenbankänderungen unabdingbaren *exactly once*-Aufrufsemantik. Diese Funktionalität, die in Abschnitt 4.2.6 bereits beschrieben ist, kann leicht in das Gateway implementiert werden.

Die Inspektion interner Tabellen-Parameter folgt einem *ad hoc*-Ansatz, der bei der Erzeugung von Tl-Iterationen und deren Übertragung in die Tabellenvariable eine Typüberprüfung für jede Tabellenzeile durchführt. Ein simpler Typ-Cache würde den Zugriff auf die dynamische Typkodierung unterbinden und damit die Marshalling- und Unmarshalling-Routinen wesentlich effizienter machen.

Die Spezifikation der Funktionsbausteinsignaturen muß in der vorliegenden Fassung zur Typprüfung und dem Parametermarshalling explizit vorgegeben werden. Auf der Seite des SAP-Applikationsservers

[2] Die RFC-Bibliothek für externe Systeme unterstützt strukturierte Parameter laut Dokumentation nicht.

sind sie jedoch bereits in der Funktionsbausteindefinition festgelegt worden. Denkbar wäre eine spezielle Funktion, die Singaturen der Funktionsbausteine aus dem SAP-Applikationsserver inferiert und damit deren dynamische Bindung vornimmt.

Funktionsbausteinabstraktion

Das SAP R/3-Gateway auf der Ebene von Funktionsbausteinen stellt einen Basismechanismus dar. Das Ziel weiterer Entwicklungen muß jedoch die Verbesserung zentraler Kriterien moderner Informationssysteme sein, wie der in der Einleitung erwähnten: Evolution, Skalierung, Verteilung und Offenheit. Dazu ist die Gateway-Konstruktion zu modernen Datenbankprogrammiersprachen, die diese Kriterien konzeptuell bereits erfüllen, ein erster Schritt, der qualitativ den Client/Server-Mechanismus des SAP R/3 zu einer neuen Entwicklungsumgebung erweitert.

Die nächste Stufe der Integration des SAP R/3 muß dessen semantische Struktur berücksichtigen. Ein erster Schritt hierfür wäre die Entwicklung spezialisierter Dienste für den Data Dictionary-Zugriff, die daß Lesen, Anlegen, Modifizieren und Löschen von Elementen der Typhierarchie und der SAP-Tabellendefinitionen zulassen, so daß eine einfache Manipulation des Schemas im SAP-Applikationsserver über einen entsprechenden abstrakten Datentyp vorgenommen werden kann. Einen weitergehenden Ansatz liefert Abschnitt 3.2.4 Seite 49, der die Basismechanismen zum Zugriff auf die semantischen Objekte des SAP R/3 vorstellt. Durch Abbildungen derselben nach Tycoon-Tl wird eine verbesserte Applikationstransparenz realisiert, mit der Möglichkeit bestehende SAP-Modul-Applikationen unter Ausnutzung vorteiliger Tycoon-Tl-Eigenschaften in neuen Kontexten[3] zu verwenden.

Implementation der neuen *Intelligent Terminal*-Schnittstelle von SAP

In Abschnitt 3.1.2 wurde die Offenheit des SAP-Systems in seinen drei Ebenen Präsentation, Applikation und Datenbank untersucht. Eine offene Schnittstelle der Präsentationsschicht SAPGUI ist das *Intelligent Terminal*. Es bietet einen algorithmengesteuerten Zugriff auf die Elemente des SAP-GUI. Dadurch können alle SAP-Transaktionen durch Automatismen gleichwertig der benutzergeführten SAP-Sitzung aufgerufen werden.

Viele der Geschäftsprozesse des SAP-Systems sind nur in Form bestimmter SAP-Transaktionen vorhanden. Durch die Verwendung des *Intelligent Terminal* wird ein modellgerechter Zugriff auf diesen durch SAP-Transaktionen realisierten Geschäftsprozessen garantiert und somit alle in den SAP-Transaktionen festgehaltenen Integritätsbedingungen automatisch respektiert. Als Vorteil ergibt sich, daß neue Informationstechnologien und moderne Wege der Benutzerinteraktion, wie sie das WWW[4], interaktive Sprachsteuerung oder Braille-Terminals für Blinde darstellen, durch die *Intelligent Terminal*-Schnittstelle einen vollwertigen interaktiven Zugriff auf die SAP-Transaktionen erhalten. Insbesondere betrifft dies auch alle Tycoon-Tl-Dienste.

[3] Mit den BAPIs strebt SAP bereits in eine ähnliche Richtung.
[4] World Wide Web

Anhang A

Glossar

4GL 4th Generation Language; Sprache der vierten Generation.

ABAP/4 Advanced Business Application Programming Language / →4GL; die Programmiersprache des SAP-Systems.

Ablauflogik Teil eines →Dynpros. Sie besteht aus den Komponenten →PBO und →PAI. Sie enthält Feldprüfungen und Aufrufe der →Module.

ALE Application Link Enabling; Konzept zur Unterstützung verteilter Anwendungen in SAP R/3. Dabei wird nicht eine gemeinsame Datenbank genutzt, sondern es werden SAP-Anwendungen, die auf verteilten Applikationsservern mit verschiedenen Datenbanken ablaufen über synchrone und asynchrone Kommunikation zu einer Gesamtanwendung integriert. Die zugehörigen Kommunikationsmechanismen sind →CPI-C, →RFC und →IDOC.

API Application Programmers Interface; eine Funktionsschnittstelle für Programmierer.

APPC Advanced Program to Program Communication; Teil des →SNA-Standards um Transaktionen über Netzwerkkommunikation auszuführen. APPC beschreibt die Schnittstelle des →LU6.2-Protokolls.

Applikation Standardanwendungen des SAP-Systems; SAP liefert mit ihrem System R/3 mehrere betriebswirtschaftliche Module aus.

BAPI Business →API; Geschäftsprozeßprogrammierschnittstellen zum →BOR des SAP R/3.

Batch Input Besonderes Eingabeverfahren für Daten in das SAP-System. Es werden dabei bestehende →SAP-Transaktionen ausgenutzt, indem auf repititive Weise →Dynpros automatisch mit Daten beschickt werden.

Berechtigungen Konzept für Benutzerrechte in SAP R/3. Dazu gehören Berechtigungs-objekte(BO), -profile(BP) und -sammelprofile(SP). Über die →Workbench werden BOs angelegt, das sind Aggregationen von Feldbeschreibungen. Über die Benutzerverwaltung können BPs angelegt und zu SPs zusammengefaßt werden. Diese werden dann bestimmten Benutzern zugewiesen. Jedes →ABAP/4-Programm kann dann durch die Anweisung `AUTHORITY-CHECK` explizit prüfen, ob ein Benutzer berechtigt ist, den Code auszuführen. Es gibt keine impliziten Prüfungen!

BOR Business Object Repository; Sammlung von Geschäftsprozeßobjekten im SAP R/3.

BPR Business Prozess Reengineering; Umstrukturierung und Optimierung der Aufbau- und Ablauforganisation eines Unternehmens.

Buchungskreis Selbständige, gemäß rechtlichen Vorschriften bilanzierende organisatorische Einheit.

Business Workflow Unterstützung der Vorgangsbearbeitung im System R/3.

CATT Computer Aided Test Tool;

CCMS Computing Center Management System; Überwachungs- und Steuerungsmechanismen für den Betrieb einer R/3-Installation.

COM Component Object Model; Microsofts Objektmodell. Wesentliche Eigenschaften sind Kapselung und Nachrichtenaustausch. COM ist die Basistechnologie für →OLE.

CPI-C Control Programming Interface for C; eine spezielle →APPC-konforme Programmierschnittstelle für die Sprache C.

CORBA Common Object Request Broker; verteiltes sprachenunabhängiges Objektmodell der OMG. Wesentliche Eigenschaften sind der entfernte Methodenaufruf und Vererbung.

CUA Common User Access; „Style Guide"der Firma IBM zur Normung von Benutzerschnittstellen und -Abläufen.

Customizing Methoden, Funktionen und Einstellungen zur Unterstützung der Einführung des Systems R/3. Insbesondere werden vorhandene Funktionen und Abläufe durch spezielle Eintragungen in Customizing-Tabellen zu einem den unternehmerischen Erfordernissen angepaßten Verhalten programmiert.

Data Dictionary Datenwörterbuch des SAP-Systems. Es enthält Typdefinitionen und Metainformationen. Die wesentlichen Objekte sind Tabellen, Domänen, Datenelemente, Matchcodes, Views, Sperrobjekte, Fremdschlüssel und Typenpools.

Datenelement Typen der Tabellenfelder. Datenelemente erweitern →Domänen um Überschriften und Dokumentationstexte. Sie stellen betriebswirtschaftlich orientierte Typen dar.

DDIC →Data Dictionary

DDL Data Definition Language; eine Sprache zur Definition von Datenbankstrukturen. Durch sie wird das Datenschema festgelegt. In SAP R/3 kann entweder über das →DDIC oder durch →Native-SQL Befehle ein Datenschema festgelegt werden.

Delta Manager Mechanismus um Datenmengen für Tabellenparameter bei →RFCs zu minimieren.

Dialogbaustein Folge von Dialogschritten eines Programms, die zu einer Modularisierungseinheit zusammengefaßt sind.

Dialogtransaktion Wird von SAP synonym mit →SAP-Transaktion verwendet.

Dispatcher Verwaltungsprozess eines Applikationsservers. Er verteilt die in einer globalen Warteschlange auflaufenden Verarbeitungsaufträge an →Workprozesse.

Domäne Typen der →Datenelemente. Domänen erweitern →externe Typen um zusätzliche Längen- und Darstellungsinformationen. Sie stellen technisch orientierte Typen dar.

Dynpro Dynamisches Programm; SAP-spezifischer Begriff für ein Bildschirmbild (→Screen)und dessen →Ablauflogik.

EDI Electronic business Data Interchange; Standartisierter Austausch von Geschäftsdaten zwischen verschiedenen Systemen mit Hilfe von vordefinierten Dokumententypen, wie Rechnung, Bestellung und Lieferschein.

EDIFACT Electronic Data Interchange For Administration, Commerce and Transport

Einführungsleitfaden SAP R/3 muß zur →Einführung in ein Unternehmen einem →Customizing unterzogen werden. Einführungsleitfäden sind interaktive Software-Komponenten des SAP R/3, die eine dem Vorgehensmodell des SAP R/3 entsprechende Einführung und damit zusammenhängendes →Customizing unterstützt.

Enqueue Service Mechanismus des Systems R/3, um die →Sperrverwaltung für betriebswirtschaftliche Objekte sicherzustellen. Der R/3 Enqueue Service unterstützt die →Sperrverwaltung in verteilten Client/Server-Umgebungen. Dazu wird ein Sperrobjekt mit zugehoerigen Funktionsbausteinen angelegt. Die Funktionsbausteine erlauben die Sperrung, Freigabe oder Pruefung. Es gibt keine impliziten Pruefungen!

Entwicklingsklasse Menge aller Entwicklungsumgebungsobjekte, die einer gemeinsamen Anwendung im Sinn des Entwicklers angehoeren.

Ereignis-Schlüsselwort ABAP/4-Schlüsselwort zur Unterstützung der externen Ablaufsteuerung.

EPK Ereignisgesteuerte Prozesskette

EU Entwicklungsumgebung; hierunter sind alle →Werkzeuge und Objekte der Workbench zu zusammengefaßt.

externer Typ Basisdatentyp der Darstellung in →Dynpros.

Feld Speicherobjekt. Variablen innerhalb von ABAP/4 sind Felder aber auch →Feldleisten aggregieren mehrere Felder.

Feldleiste Speicherobjekt dessen Typ durch eine →Struktur bestimmt wird. Feldleisten können „first class“-Objekte der Sprache ABAP/4 sein oder Zeilen von Datenbanktabellen.

Fremdschlüssel Feldreferenz einer Tabelle auf Felder einer anderen Tabelle. Durch die Angabe von Einschränkungen des Wertebereichs der referenzierenden Felder auf Werte, die in Feldleisten der referenzierten Tabelle auftreten, wird die *referentielle Integrität* sichergestellt.

Funktionsbaustein ABAP/4-Programm mit klar definierter Schnittstelle. Sie Schnittstelle enthält Eingabe-, Ausgabe- und Tabellenparameter, sowie alle Ausnahmen, die auftreten können.

Funktionsbibliothek Die Menge aller →Funktionsbausteine eines SAP-Applikationsservers.

Funktionsgruppe Logisch oder anwendungstechnisch zusammengehörige →Funktionsbausteine.

IDOC Intermediate Document; Austauschformat fuer betriebswirtschaftliche Daten in SAP R/3.

Instanz Administrative Einheit von Komponenten des R/3 Systems. Zu ihr gehört genau ein →Dispatcher und verschiedene →Workprozesse. Alle Komponenten einer Instanz lesen ein gemeinsames Profil. Ihr Name ergibt sich aus den Kürzeln der Workprozess-Dienste, an die noch eine Instanznummer angehängt wird. Beispiel: DVM01 für Dialog-, Verbuchungs- und Message-Dienst der Instanz Nummer 01.

IMG Implementation Guide; im System R/3 enthaltene →Einführungsleitfäden zur Einstellung (→Customizing) der Anwendungen bei der Einführung.

Kopfleiste Cursor-Bereich einer Datenbanktabelle oder einer internen Tabelle.

KTW Korrektur und Transportwesen; Methoden und Werkzeuge des SAP R/3, die die Teamentwicklung, die Versionierung und den Transport von →EU-Objekten unterstuetzen.

Logische Datenbank Zusammenfassung mehrerer hierarchisch ueber Fremdschluessel verbundener Tabellen. Reportprogramme koennen auf logischen Datenbanken durch ausschliessliche Angabe bestimmter →Ereignisschluesselwoerter ablaufen.

LU6.2 Logical Unit Version 6.2; SNA-Kommunikationprotokoll.

LUW Logical Unit of Work; Transaktion in SAP R/3, die durch `COMMIT-WORK` abgeschlossen oder durch `ROLLBACK-WORK` zurueckgesetzt wird.

Mandant Handelsrechtlich, Organisatorisch und datentechnisch selbständige Einheit im System R/3. Realisiert werden Mandanten durch Tabellen, die als erstes Feld ein Mandanteneintrag vom externen Typ `CLNT` haben.

Matchcode Sekundärer Zugriffspfad auf Daten, der in Dialogen eine schnelle interaktive Suche nach bestimmten Informationen unterstützt. Matchcodes werden direkt im →DDIC angelegt.

Menu Painter Werkzeug der →Workbench, das die Gestaltung von Menus und Button-Leisten ermoeglicht.

Modulpool →ABAP/4-Programm, daß im wesentlichen →Module enthält.

Modul Unterroutine für Aufrufe aus der →Ablauflogik. Ein Modul enthält den →ABAP/4-Code, der die eigentliche Datenverarbeitung bei →Dialogtransaktionen durchführt.

Modus Unabhängiger Terminalemulator einer Benutzersitzung. Es können bis zu sechs Modi parallel offen sein, wenigstenz aber einer pro Sitzung.

Native-SQL ABAP/4-Schnittstelle zum SQL-Dialekt der benutzten Datenbank. Durch ihre Verwendung wird ein ABAP/4-Programm Datenbankabhaengig.

OCX OLE Custom Control;

Open-SQL SAP eigener in ABAP/4 integrierter SQL-Dialekt. Er wird durch die Datenbankschnittstelle in den SQL-Dialekt der speziell benutzten Datenbank uebersetzt. Ihre Verwendung garantiert Offenheit gegenueber allen Datenbanken, die von SAP unterstuetzt werden.

OLE Object Linking and Embedding; Microsofts proprietäre Norm zur Konstruktion von Dokumenten und Anwendungen aus wiederverwendbaren Komponenten. Sie basiert seit Version 2 vollständig auf →COM.

PAI Process After Input; Teil der →Ablauflogik für die Verarbeitung nach einer Benutzereingabe in einem →Dynpro.

PBO Process Before Output; Teil der →Ablauflogik für die Verarbeitung vor einer Benutzereingabe in einem →Dynpro.

Pretty Printer Funktion des ABAP/4-Editors, um Quelltext einheitlich zu formatie5ren.

PUT Programming Update Tool. Eigenstaendiges Programm zur Durchfuehrung eines Versionswechsels in einem SAP-System.

Q-API Queue Application Programmers Interface. →Funktionsgruppe fuer die Durchfuehrung asynchroner Kommunikation auf Basis des synchronen →RFCs.

Report Ausführbares Programm für die Listenerzeugung mit vorgegebenen Parametern.

Repository Persistenter Speicher aller langlebigen SAP-Daten. Dazu gehören das →Data Dictionary, →Programme, →Dynpros und transparente Tabellen.

R/3 Realtime System Version 3. Zusätzlich wird noch eine weitere Versionsnummer vergeben. Die vorliegende Arbeit wurde mit SAP R/3 Version 2.2j begonnen und basiert in ihrer entgültigen Form auf SAP R/3 Version 3.0c.

RFC Remote Function Call; entfernter Funktionsaufruf mit Kontrollflußsteuerung und Parametermarshalling.

SAP Systeme, Anwendungen und Programme in der Datenverarbeitung; Firmenname.

SAPcomm Kommunikationsserver des SAP-Systems. Er unterstützt nachrichtenbasierte Kommunikation, wie z.B. E-Mail.

SAPGUI SAP Graphical User Interface; Eigenständiges Programm zur Benutzerinteraktion mit dem R/3-System.

SAP-Transaktion Folge von Dialogschritten(→Dynpro)

SNA System Network Architecture; Netzwerknorm der IBM.

Screen Eine Bildschirmdarstellung des SAP-Terminalemulators.

Screen Painter Werkzeug der →Workbench, mit dem Dynpros angelegt, modifiziert, angezeigt und gelöscht werden.

Sperrverwaltung →Enqueue-Service

Struktur Aggregation aus mehreren Datentypen. Abhängig vom Kontext sind das →Datenelemente für Strukturen die im →Data Dictionary deklariert werden oder ABAP/4-Typen bei Strukturen, die innerhalb eines ABAP/4 Quelltextes deklariert werden.

Tabelle Einziger Mengendatentyp des SAP-Systems. Eine Tabelle enthält →Feldleisten, die sich ihrerseits aus →Feldern zusammensetzt. Im SAP-System werden folgende Typen unterschieden:

interne Tabelle Sie nimmt einen Sonderstatus ein. Sie ist nicht persistent und dient dazu zur Laufzeit mengenwertige Ergebnisse aufzunehmen und zu verarbeiten.

Pool Tabellenaggregat.

Cluster Tabellenaggregat

Transp Transparente Datenbanktabelle. Solche Tabellen werden 1:1 auf DB-Tabellen abgebildet, wobei die externen Typen des SAP-Systems durch die DB-Schnittstelle auf DB-Typen des unterliegenden RDBMs abgebildet werden.

Transaktion Im Kontext von SAP ist eine Folge von →Dynpros gemeint. Nicht verwechseln mit Datenbanktransaktionen. In dieser Arbeit heissen sie →SAP-Transaktionen.

Verbuchung Durchführung einer Datenbankänderung.

Views (Datenbank, Help, Streich, Strukturen, Customizing)

Vorgehensmodell Vier-Phasenmodell des SAP R/3, daß die Grundlage für die Neueinführung des Systems in Unternehmen.

Work Prozess(WP) Betriebssystemprozess, dem eine spezifische Aufgabe zugeordnet ist. Die (Dialog, Verbuchung, Enqueue, Batch, Message, Gateway, Spool)

Workbench Die Entwicklungsumgebung des SAP-Systems. Zu ihr gehoeren alle Entwicklungswerkzeuge: →Screen-Painter, →Menu-Painter, →Funktionsbibliotheksverwaltung, →ABAP/4-Editor.

Literaturverzeichnis

[ABB96] Bc abap/4 – basics. In *ABAP/4 User's Guide*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[ALE96] SAP AG. *CA ALE Application Link Enabling*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[AOA96] Bc abap/4 ole automation controller. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[BAP97] SAP AG, Neurottstraße 16, 69190 Walldorf, Germany. *The BAPIs in R/3 3.1*, 1997.

[BBC+96] Thomas Barthel, Andreas Blume, Ingmar Carlberg, Michael Kühn, Reinhard Linz, Brigitte Maschmann-Schulz, and Georg Siebert. Was ist sap? In *Gibt es Alternativen zu SAP?*, pages 49–52. Verlag Vieweg, Braunschweig/Wiesbaden, 1996. ISBN 3-528-05536-7.

[BEG95] Rüdiger Buck-Emden and Jürgen Galimow. *Die Client/Server-Technologie des Systems R/3*. Addison Wesley Verlag, Bonn Paris, zweite edition, 1995.

[BEG96] Rüdiger Buck-Emden and Jürgen Galimow. *Die Client/Server-Technologie des Systems R/3*. Addison Wesley Verlag, Bonn Paris, dritte edition, 1996. ISBN 3-89319-764-8.

[BG82] Friedrich L. Bauer and Gerhard Goos. *Informatik – Eine einführende Übersicht*, volume eins. Springer-Verlag, Berlin Heidelberg New York Tokyo, dritte edition, 1982. ISBN 3-540-11722-9.

[BG84] Friedrich L. Bauer and Gerhard Goos. *Informatik – Eine einführende Übersicht*, volume zwei. Springer-Verlag, Berlin Heidelberg New York Tokyo, dritte edition, 1984. ISBN 3-540-13121-3.

[BL84] R. Burstall and B. Lampson. A kernel language for abstract data types and modules. In *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*. Springer-Verlag, 1984.

[Car90] Luca Cardelli. The quest language and system(tracking draft). Technical report, Digital Systems Research Center,DEC SRC, Palo Alto, 1990. (shipped as part of the Quest V.12 system ditribution).

[CCo96] SAP AG. *BC SAP Communication: Configuration*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[CDK94] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems*. Addison-Wesley, second edition, 1994. ISBN 0-201-62433-8.

[CMMS91] L. Cardelli, S. Martini, J.C. Mitchell, and A. Scedrov. An extension of system f with subtyping. In T. In Ito and A.R. Meyer, editors, *Theoretical Aspects of Computer Software, TACS'91*, Lecture Notes in Computer Science, pages 750–770. Springer Verlag, 1991.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Communicating of the ACM*, 13:377–387, June 1970.

[Com91] Douglas E. Comer. *Internetworking with TCP/IP*, volume one. Prentice Hall International, 1991. ISBN 0-13-474321-0.

[CPG96] SAP AG. *BC SAP Communication: Programmer's Guide*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[Dat95] C. J. Date. *An Introduction to Database Systems.* Addison-Wesley Publishing Company Inc., sixth edition, 1995. ISBN 0-201-82458-2.

[DDI96] SAP AG. *BC ABAP/4 Dictionary*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[DM92] William Davidow and Micheal Malone. *The virtual corporation.* HarperCollins, New York, July 1992.

[Fri91] Linda Weiser Friedman. *Comparative Programming Languages.* Prentice-Hall International Editions, 1991. ISBN 0-13-155714-9.

[FRJL88] Charles N. Fisher and Jr. Richard J. LeBlanc. *Crafting A Compiler.* The Benjamin/Cummings Publishing Company, Inc., 2727 Sand Hill Road, Menlo Park, California 94025, 1988. ISBN 0-8053-3201-4.

[Gei95] Andreas Geisler. Basisdienste zur gestaltung einer reflektiven grafischen entwicklungsumgebung für eine persistente programmiersprache. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, June 1995.

[Göl96] Martin Göllnitz. Studienarbeit: Polymorphe, persistente client/server-programmierung mit dynamischer, hierarchischer adressauflösung. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, April 1996.

[Gor95] Lukas Gorzynski. Studienarbeit: Entfernter datenzugriff auf persistente objektspeicher unter benutzung standardisierter protokolle (tsp/nfs). Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, October 1995.

[GR93] Jim Gray and Andreas Reuter. *Transaction Processing: Concepts and Techniques.* Morgan Kaufmann Publishers, san Mateo, California, 1993. ISBN 1-55860-190-2.

[IBM88] *Systems Applications Architecture Common Programming Interface, Communications Reference*, May 1988. SC26-4399-0.

[IDO96] SAP AG. *CA The IDOC Interface for Electronic Data Interchange (EDI)*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[Joh95] Nico Johannisson. Studienarbeit: Generische programmierung typsicherer kommunikationsdienste. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, May 1995.

[JV87] Eike Jessen and Rüdiger Valk. *Rechensysteme.* Springer Verlag, Berlin Heidelberg, 1987. ISBN 3-540-16383-2.

[Kel96] Kaliana Kellye. Sap-faq. http://www.informatik.Uni-Oldenburg.de, September 1996. news:de.alt.comp.sap-r3.

[KK71] W. Klingenberg and P. Klein. *Lineare Algebra und Analytische Geometrie*, volume 1. Bibliographisches Institut AG, Mannheim, 1971. ISBN 3-411-00748-6.

[Koc96] Christian Koch. Studienarbeit: Generisches und typsicheres modulmanagement. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, March 1996.

[Koe96] Hubertus Koehler. Generische daten- und funktionsvisualisierung in einer persistenten programmierumgebung. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, March 1996.

[Kor93] Marcel Kornacker. Studienarbeit: Eine polymorph typisierte sql-schnittstelle. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, April 1993.

[Kor95] Marcel Kornacker. Persistente sicherungspunkte für langlebige aktivitäten in offenen umgebungen. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, August 1995.

[LD87] Peter C. Lockemann and Klaus R. Dittrich. Architektur von datenbanksystemen. In In P.C. Lockemann and J.W. Schmidt, editors, *Datenbank-Handbuch*. Springer Verlag, Berlin Heidelberg New York, 1987. ISBN 3-540-10741-X.

[Mat93] F. Matthes. *Persistente Objektsysteme: Integrierte Datenbankentwicklung und Programmerstellung*. Springer-Verlag, 1993.

[Mat96a] Bernd Mathiske. *Mobilität in persistenten Objektsystemen*. PhD thesis, Fachbereich Informatik, Universität Hamburg, Germany, October 1996.

[Mat96b] Bernd Matzke. *ABAP/4 Die Programmiersprache des SAP-System R/3*. Addison-Wesley GmbH, Deutschland, erste edition, 1996. ISBN 3-89319-949-7.

[Men96] Ulrich Mende. Automatisierte schnittstellenentwicklung für r/3. *IX Multiuser Multitasking Magazin*, 10:128–142, Oktober 1996.

[MME96] SAP AG. *BC Memory Management*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[MMM93] B. Mathiske, F. Matthes, and S. Müßig. The tycoon system and library manual. DBIS Tycoon Report 212-93, Fachbereich Informatik, Universität Hamburg, Germany, December 1993. (Revised 19-jan-1996).

[MMS94] F. Matthes, S. Müßig, and J.W Schmidt. Persistent polymorphic programming in tycoon: An introduction. FIDE Technical Report Series FIDE/94/106, FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ, August 1994.

[MMS95] F. Matthes, R. Müller, and J.W. Schmidt. Towards a unified model of untyped object stores: Experience with the Tycoon store protocol. In M.P. Atkinson, editor, *Fully Integrated Data Environments*. Springer-Verlag (to appear), 1995.

[MS92] F. Matthes and J.W. Schmidt. Definition of the tycoon language - a preliminary report. Informatik Fachbericht FBI-HH-B-160/92, Fachbereich Informatik, Universität Hamburg, Germany, October 1992. (Revised 17-aug-1995).

[MS93] F. Matthes and J.W. Schmidt. System construction in the tycoon environment: Architectures, interfaces and gateways. In *Proceedings of Euro-Arch'93 Congress*. Springer-Verlag, October 1993.

[MS94a] F. Matthes and J.W. Schmidt. Persistent threads. FIDE Technical Report Series FIDE/94/88, FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ, August 1994. (A shorter version of this text appeared as [MaSc94].).

[MS94b] F. Matthes and J.W. Schmidt. Persistent threads. In *Proceedings of the Twentieth International Conference on Very Large Data Bases, VLDB*, pages 403–414, Santiago, Chile, September 1994. (An extended version of this text appeared as [MaSc94b]).

[MS97] F. Matthes and J. W. Schmidt. Datenbankmodelle und datenbanksprachen. Datenbankhandbuch in Vorbereitung, 1997.

[Nie92] Claudia Niederée. Generische dienste für datenintensive anwendungen: Iterationsabstraktion, integritätsüberwachung, fehlererholung. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, November 1992.

[Nie96] Oliver Nietsch. Studienarbeit: Die anbindung des tycoon-systems an die case-umgebung stp über ein repository. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, March 1996.

[Nie97] Oliver Nietsch. Modellierung und validierung dynamischen systemverhaltens am beispiel von tycoon und stp/omt. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, April 1997.

[Nut92] Gary J. Nutt. *Open Systems*. Prentice Hall Inc., 1992. ISBN 0-13-636234-6.

[OHE96a] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Client/Server Survival Guide*. John Whiley & Sons, Inc., New York Chichester Brisbane Toronto Singapore, second edition, 1996. ISBN 0-471-15325-7.

[OHE96b] Robert Orfali, Dan Harkey, and Jeri Edwards. *The Essential Distributed Objects Survival Guide*. John Whiley & Sons, Inc., New York Chichester Brisbane Toronto Singapore, 1996. ISBN 0-471-12993-3.

[OLE96a] Bc ole programming for r/3 desktop integration. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[OLE96b] Bc ole programming with business objects. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[PCWT96] Norman Paton, Richard Cooper, Howard Williams, and Philip Trinder. *Programmiersprachen für Datenbanken – SELECT LIKE WHERE FROM*. Prentice Hall, München London Mexiko New York Singapur Sydney, 1996. ISBN 3-8272-9527-0.

[Pou96] Nastaran Vaziri Pour. Studienarbeit: Flexible bindungstechniken für ubiquitäre ressourcen in verteilten anwendungen. Master's thesis, Fachbereich Informatik, Universität Hamburg, Germany, March 1996.

[Raa91] Jörg Raasch. *Systementwicklung mit Strukturierten Methoden*. Carl Hanser Verlag München Wien, 1991. ISBN 3-446-16147-3.

[RBP+93] J. Rumbaugh, M. Blaha, W. Premerlami, F. Eddy, and W. Lorensen. *Objektorientiertes Modellieren und Entwerfen*. Carl Hanser Verlag, München Wien, 1993. ISBN 3-446-17520-2.

[Reu87] A. Reuter. Maßnahmen zur wahrung von sicherheits und integritätsbedingungen. In In P.C. Lockemann and J.W. Schmidt, editors, *Datenbank-Handbuch*. Springer Verlag, Berlin Heidelberg New York, 1987. ISBN 3-540-10741-X.

[RFC96a] Bc the rfc api. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[RFC96b] Bc the rfc generator. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[RFC96c] Bc rfc programming in abap/4. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[SAP94a] SAP AG, Neurottstraße 16, 69190 Walldorf, Germany. *Funktionen im Detail – ABAP/4 DEVELOPMENT WORKBENCH*, September 1994.

[SAP94b] SAP AG, Neurottstraße 16, 69190 Walldorf, Germany. *Funktionen im Detail – SAP R/3 SOFTWAREARCHITEKTUR*, Juni 1994.

[SAP96a] SAP AG, SAP. *R/3 Online Dokumentation*, release 3.0d edition, Januar 1996. CDROM.

[SAP96b] SAP AG, SAP. *R/3 Online Handbuch*, release 3.0 edition, März 1996. interaktive Hilfe.

[SAP96c] Sap – world leader in enterprise business solutions / a corporate profile. WWW: http://www.sap-ag.de/discsap/annual/report.htm, June 1996.

[Sch87] J.W. Schmidt. Datenbankmodelle. In P.C. Lockemann and J.W. Schmidt, editors, *Datenbank-Handbuch*. Springer Verlag, Berlin Heidelberg New York, 1987. ISBN 3-540-10741-X.

[Sch95] Hans-Jürgen Scheruhn. Standardsoftware-einführung mit dem aris-toolset – modellbau. *IX Multiuser Multitasking Magazin*, 4:160–170, April 1995.

[SM93] J.W. Schmidt and F. Matthes. Lean languages and models: Towards an interoperable kernel for persistent object systems. In *Proceedings of the IEEE International Workshop on Research Issues in Data Engineering RIDE '93 Vienna, Austria*, pages 2–17, April 1993. Also appeared as TR FIDE/93/59, FIDE Technical Report Series, FIDE Project Coordinator, Department of Computing Sciences, University of Glasgow, Glasgow G128QQ.

[Ste92] W. Richard Stevens. *Programmieren von UNIX-Netzen*. Prentice Hall, 1992. ISDN 0-13-928748-5.

[Ste96] Ulrike Steffens. Integration von information retrieval funktionalität in eine offene, persistente programmierumgebung. Informatik Mitteilung FBI-HH-M-257/96, Fachbereich Informatik, Universität Hamburg, Germany, April 1996. (Master's Thesis).

[Tan95] Andrew S. Tanenbaum. *Verteilte Betriebssysteme*. Prentice Hall Verlag GmbH, 1995. ISBN 3-930436-23-X.

[TSC96] SAP AG. *BC The SAP Communications Server*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[VGH93] Gottfried Vossen and Margret Groß-Hardt. *Grundlagen der Transaktionsverarbeitung*. Addison-Wesley, Bonn Paris, 1993. ISBN 3-89319-576-9.

[Vos95] Gottfried Vossen. *Datenbank Theorie*. International Thomson Publications, 1995. ISBN 3-8266-0126-2.

[VRF96] Bc visual rfc service provide: Users guide. In *Remote Communications*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[WAR96] Bc writing abap/4 reports. In *ABAP/4 User's Guide*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[WAT96] Bc writing abap/4 transactions. In *ABAP/4 User's Guide*, SAP R/3 Printdoku. SAP AG, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[WBT96a] SAP AG. *BC ABAP/4 Workbench Tools*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[WBT96b] SAP AG. *BC ABAP/4 Workbench Tutorial*, SAP R/3 Printdoku, Neurottstraße 16, 69190 Walldorf, Germany, 1996. CDROM.

[Wen95a] Wenzel. *Einführung von SAP-R/3 Band 1: Projektstudien, Grundlagen und Anregungen zum Arbeiten mit SAP-R/3*, volume one. Vieweg Verlag, 1995.

[Wen95b] Wenzel. *Geschäftsprozessoptimierung mit SAP R/3: Modellierung, Steuerung und Management betriebswirtschaftlich integrierter Geschäftsprozesse*. Vieweg Verlag, 1995.

[Wen96] Paul Wentzel. *Betriebswirtschaftliche Anwendungen des integrierten Systems SAP R/3*. Vieweg Verlag, zweite edition, 1996. ISBN 3-528-15509-4.

[WHSH95] Liane Will, Christiane Hienger, Frank Straßenburg, and Rocco Himmer. *R/3-Administration*. Addison-Wesley Publishing Company, 1995. ISBN 3-89319-967-5.

[Woh88] Dirk Wohlberg. Advanced progam to program communication(appc). Studienarbeit, Universität Hamburg / Fachbereich Informatik, Mai 1988.

[Zeh85] C. A. Zehnder. *Informationsysteme und Datenbanken*. Teubner, Stuttgart, dritte edition, 1985. ISBN 3-519-02480-2.