Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Datenbanken und Informationssysteme

Studienarbeit

Die Anbindung des Tycoon-Systems an die CASE-Umgebung StP über ein Repository

Betreuer:
Prof. Dr. J. W. Schmidt
Gerald Schröder

vorgelegt von:
Oliver Nietsch
Große Straße 56 b
21075 Hamburg
Tel.: 040 - 792 63 55

Hamburg, den 19. März 1996

Typographie

In dieser Arbeit werden unterschiedliche Schriftarten verwendet, um die Lesbarkeit zu verbessern. Die Bedeutung der einzelnen Schriftarten ist in der folgenden Tabelle wiedergegeben:

Schriftart	**Bedeutung**	**Beispiel**
Emphasized	englischer Begriff	*Software Engineering*
Sans Serif	Firmen- oder Produktname	Software through Pictures T Tool
`Typewriter`	Systemkommandos oder C-Ausdrücke	`LD_LIBRARY_PATH=...` `char *sig = "Name"`
Slanted	Tycoon Language Ausdücke	*omsRepository.check(rep)*
Slanted	Tycoon Language reserviertes Schlüsselwort	***let*** a = *3*

Für die Darstellung der Tycoon Language (TL) Implementierungsdetails gelten die gültigen Tycoon *Layout*- und Namenskonventionen wie in [Mathiske et al. 93] angegeben.

Inhaltsverzeichnis

1. Einleitung

Die heutige professionelle Softwareerstellung ist dadurch gekennzeichnet, daß eine Entwicklungsmethode ohne die Unterstützung durch computerbasierte Werkzeuge kaum noch durchführbar beziehungsweise handhabbar ist. Diese Problematik ist um so stärker ausgeprägt, je komplexer der Softwareentwicklungsprozeß sich darstellt. Die Nachfrage nach immer leistungsfähigeren Anwendungssystemen erfordert daher stetige Verbesserungen im eigentlichen Prozeß der Systementwicklung, sowohl in quantitativer als auch in qualitativer Hinsicht.

Innerhalb des *Computer Aided Software Engineering* (CASE) versucht man, diesen Anforderungen durch den Einsatz geeigneter Softwarewerkzeuge nachzukommen [Balzert 93]. Die Integration der CASE-Werkzeuge in den Anwendungserstellungsprozeß erfolgt auf der Basis einer gemeinsamen Entwicklungsdatenbank, dem *Repository*, welches als ein zentraler Bestandteil einer computerunterstützten Entwicklungsumgebung gilt.

Um flexibel, problemadäquat und sicher auf die hohe Anzahl der von den CASE-Werkzeugen angelegten persistenten Modellierungsobjekten in der Entwicklungsdatenbank zugreifen zu können, ist eine geeignete Klasse von Softwaresystemen erforderlich. Zur Entwicklung solcher Systeme, der sogenannten persistenten Objektsysteme, bietet das Tycoon[1]-System als integrierte Programmierumgebung am Arbeitsbereich DBIS der Universität Hamburg einen geeigneten sprachlichen und architektonischen Rahmen, der unter anderem Konzepte der Erweiterbarkeit, Offenheit, Modularität, Typisierung und Polymorphie umfaßt [Matthes 93].

1.1 Motivation

Den Hintergrund dieser Arbeit bildet die im Kapitel 2 beschriebene CASE-Umgebung Software through Pictures (StP), auf deren Basis die implementierte Softwareentwicklungsmethode OMT [Rumbaugh et al. 91] unter anderem sogenannte *Harel State Charts* enthält. Hierbei handelt es sich um Zustandsautomaten zur Prozeßbeschreibung, die dem Anwendungsentwickler im Rahmen der dynamischen Systemmodellierung als Ergänzung zu Beschreibungsmethoden des statischen Systemaufbaus dienen. Die persistente Ablage der grafischen Beschreibung dieser Zustandsdiagramme erfolgt im StP Repository.

[1] Tycoon: Typed Communicating Objects in open Environments

Aufbauend auf dieser Arbeit soll auf der Tycoon-Ebene eine Modellierungs- und Simulationsbibliothek entwickelt werden, welche für die Systementwicklung Werkzeuge zur Animation und Simulation der auf der StP/OMT Seite grafisch entwickelten Prozeßbeschreibungen zwecks Entwurfsvalidierung bietet.

Diese Animationskomponente soll in die dem Softwareentwicklungsprozeß zugrundeliegende Entwurfsmethode integriert werden. Somit kann die Simulation des dynamischen Systemverhaltens als Ergänzung zur statischen und dynamischen Modellierung von Anwendungssystemen gelten. Eine Erweiterung des Werkzeugs auf die Simulation, Animation und Implementierung von *Workflow*-Modellen ist ebenfalls geplant.

1.2 Problemstellung

Eine Voraussetzung für die Realisierung der Animationskomponenten auf der Tycoon-Ebene ist die Möglichkeit zur Übernahme der gesamten, im StP Repository abgelegten Modellierungsdaten auf die Tycoon-Ebene.

Inhalt dieser Arbeit ist, Zugriffsmöglichkeiten auf Objekte der Entwicklungsdatenbank zu erkunden und zu bewerten, inwieweit diese unter dem Blickwinkel des zu realisierenden Repositoryzugriffs aus dem Tycoon-System heraus benutzbar sind.

Desweiteren soll auf der Tycoon-Ebene eine Bibliothek erstellt werden, die den Zugriff auf alle Repositoryobjekte bietet. Diese Umgebung ist hinsichtlich unterschiedlicher Kriterien wie zum Beispiel Flexibilität, Typsicherheit, Wartbarkeit etc. zu bewerten. Außerdem sollen die der Implementierung zugrundeliegenden Aspekte dokumentiert und erläutert werden.

1.3 Struktur der Arbeit

Dieser Einleitung anschließend folgt im Kapitel 2 ein kurzer Überblick über computerunterstützte Anwendungsentwicklung vor dem Hintergrund der Benutzung von Software through Pictures und die Integration von CASE-Werkzeugen innerhalb der StP Umgebung. Desweiteren werden Dienste und Merkmale der CASE-Umgebung StP dargestellt.

Kapitel 3 vermittelt eine Übersicht über das StP Repository. Eine besondere Betrachtung erfährt das Objektmanagement. Hierbei ist das Informationsmodell, das *Persistent Data Model* (PDM), nach dem das Repository implementiert ist, für die Typisierung der StP Bibliothek auf der Tycoon-Ebene von Bedeutung. Ferner wird betrachtet, wie eine Abbildung von Anwendungsdatentypen auf Repositorytypen erfolgt.

Im Kapitel 4 werden werkzeugunabhängige Zugriffsmöglichkeiten auf Objekte der StP Entwicklungsdatenbank exemplarisch umgesetzt und anschließend bewertet. Dabei handelt es sich um ein StP Query and Reporting Language (QRL) Skript und ein C-Programm, in welches Funktionen des StP Application Program Interface (API) integriert sind.

Die Anbindung des Tycoon-Systems an die StP Umgebung wird im Kapitel 5 beschrieben. Dabei wird das Tycoon-System als eine in das StP Produktumfeld integrierte Entwicklungsumgebung dargestellt. Der funktionale Umfang der Realisierung des Repositoryzugriffs wird aufgeführt, modelliert und auf Tycoon-Funktionalität abgebildet.

Anschließend wird im Kapitel 6 eine Bewertung der entwickelten Tycoon-Anbindung an die StP Umgebung vorgenommen. Der Zugriff aus dem Tycoon-System auf Repositoryobjekte wird hinsichtlich unterschiedlicher Kriterien bewertet.

Diese Arbeit findet ihren Abschluß im Kapitel 7 mit einer Zusammenfassung und einem Ausblick auf eventuell noch zu berücksichtigende Aspekte in Zusammenhang mit dieser Untersuchung.

2. Die CASE-Umgebung Software through Pictures (StP)

Werkzeuge des *Computer Aided Software Engineering* sollen, neben den im Kapitel 1 erwähnten Verbesserungen, zu einer Optimierung des Prozesses der Softwareentwicklung führen, um somit unter anderem die Transparenz und Kommunikation innerhalb der Produktentwicklung zu erhöhen. Dabei hat sich die Bedeutung des Begriffs der computerunterstützten Softwareentwicklung im Verlauf der vergangenen Jahre verändert. Verstand man früher unter automatisierter Unterstützung isolierte Softwarewerkzeuge, so ist es heute eine integrierte Anzahl von Werkzeugen, mit denen alle Phasen des Softwarelebenszyklus umgesetzt werden können.

Software through Pictures (StP) der Firma Interactive Development Environments (IDE) entspricht diesem Ansatz integrierter CASE-Umgebungen. Bei StP handelt es sich um eine Familie integrierter Mehrbenutzer CASE-Umgebungen zur Entwicklung neuer und zum *Re-Engineering* bestehender Systeme.

Jedes StP Produkt beinhaltet eine Familie integrierter Werkzeuge, zum Beispiel, wie in der Abbildung 2.2 skizziert, grafische Editoren für die Systemmodellierung, Systeme zur Dokumenterstellung, Schnittstellen zu Versionskontrollsystemen etc., wobei die Grafikeditoren anerkannte Methoden der Softwareentwicklung unterstützen. Desweiteren benutzen alle StP Produkte eine gemeinsame Entwicklungsdatenbank, das im Kapitel 3 erläuterte StP Repository.

2.1 Anforderungen an eine computerunterstützte Anwendungsentwicklung

Grundlage einer automatisierten Anwendungsentwicklung ist nicht allein die Leistungsfähigkeit der einzelnen CASE-Werkzeuge. Vielmehr müssen die eingesetzten Werkzeuge im Kontext mit der ihnen zugrundeliegenden CASE-Umgebung betrachtet werden. Dabei lassen sich die Anforderungen je nach Kategorie der eingesetzten Computerunterstützung trennen. Nach allgemeinen Anforderungen, die an eine CASE-Umgebung gestellt werden, und

solchen, die von den einzelnen Werkzeugen dieser Umgebung zu erfüllen sind [Balzert 93]. Für diese Arbeit relevante Anforderungen umfassen:

Basisanforderungen an die CASE-Umgebung:

- Integrierte Datenhaltung aller Datenbasen jedes separaten CASE-Werkzeugs zur Bildung von Werkzeugketten
- Erweiterbarkeit um weitere Werkzeuge
- Offenheit und Transparenz bezüglich der Export- und Importschnittstellen
- Multi- und Interprojekt-Fähigkeit
- Portabilität (wegen Langlebigkeit und hoher Investitionskosten)

Anforderungen an einzelne CASE-Werkzeuge:

- Export- und Importschnittstellen zur Kommunikation mit anderen Werkzeugen oder der CASE-Umgebung
- Integrierbarkeit in die CASE-Umgebung

Neben diesen technisch bedingten Anforderungen müssen die eingesetzten Werkzeuge natürlich auf die der CASE-Umgebung zugrundeliegende Methode der Anwendungsentwicklung abgestimmt sein, um überhaupt eine durchgängige Werkzeugkette über alle Phasen des Softwareerstellungsprozesses gewährleisten zu können.

2.2 Werkzeugintegration

Eine effektive Unterstützung des Anwendungsentwicklungsprozesses kann erfolgen, indem sämtliche verwendeten separaten Werkzeuge in einer integralen Art miteinander verknüpft werden, so daß ihre Kombination ein einziges, großes Gesamtwerkzeug, nämlich die komplette Entwicklungsumgebung, ergibt. Die Integrationsbeziehungen zwischen den einzelnen Werkzeugen beziehen sich auf die Präsentation der Mensch-Werkzeug-Schnittstelle (*Presentation Integration*), die von allen Werkzeugen verwendeten Daten (*Data Integration*), den Kontrollfluß zwischen den Werkzeugen innerhalb des Entwicklungsprozesses (*Control Integration*) und den Prozeß des effektiven Einsetzens der Werkzeuge selbst (*Process Integration*) [Thomas, Nejmeh 92]. Die Integration aller Werkzeuge einer verteilten Anwendungsentwicklungsumgebung findet ihre Berücksichtigung in der Plattformintegration.

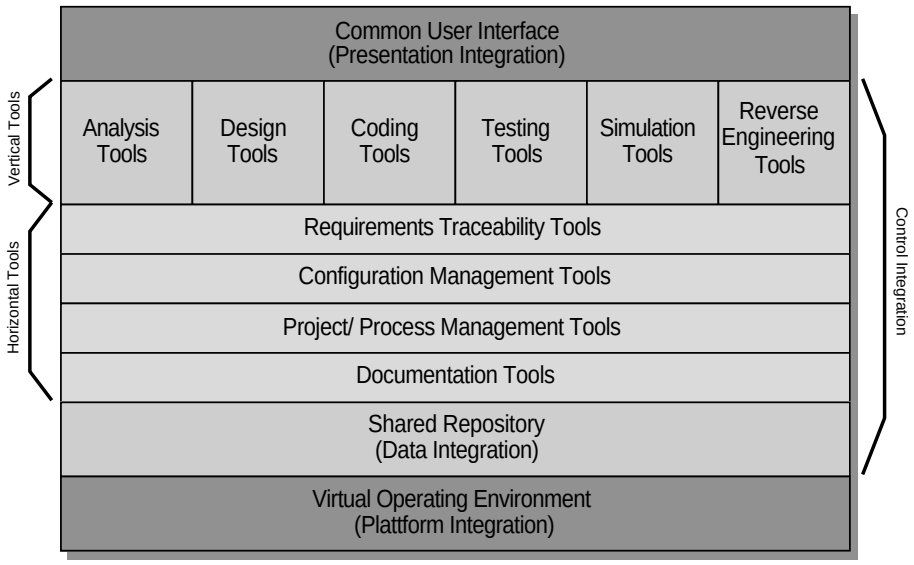

Abbildung 2.1: Integrierte CASE-Umgebung

In der Abbildung 2.1 sind die erwähnten Integrationsbeziehungen zwischen Werkzeugen einer computerunterstützten, verteilten Entwicklungsumgebung dargestellt [IDEStPPC 93]. Wichtig ist hierbei, daß die Integration aller Werkzeuge im Rahmen der Datenintegration über das Repository erfolgt. Auf die Entwicklungsdatenbank greifen sowohl die den Entwicklungsprozeß in den jeweiligen Phasen unterstützenden Werkzeuge (*Vertical Tools*), als auch die phasenübergreifenden Werkzeuge (*Horizontal Tools*) zu.

Software through Pictures Umgebungen realisieren eine offene Architektur, da die Werkzeugkommunikation über die Entwicklungsdatenbank unter Berücksichtigung von Netzwerk-, Druck- und Benutzeroberflächenstandards erfolgt. Die offene Architektur ermöglicht eine Erweiterung oder Individualisierung der StP Umgebung.

2.3 Basisdienste des StP Kerns

Alle Produkte der StP Familie sind oberhalb eines methodenunabhängigen Kerns angesiedelt. Dieser leistet die im Abschnitt 2.1 geforderten, grundlegenden Dienste der integrierten StP Entwicklungsumgebung, auf welche die jeweiligen CASE-Werkzeuge zugreifen. In unterschiedliche Kategorien zusammengefaßt, handelt es sich konkret bei diesen Diensten um das Desktopmanagement, die Prozeßmodellierung, die Versionskontrolle und das Konfigurationsmanagement, die Mehrbenutzerkoordination, die Systemindividualisierung und Erweiterbarkeit sowie die optimale Visualisierung komplexer Diagramme.

Für die Realisierung dieser Arbeit sind drei weitere Dienste des StP Kerns näher zu betrachten:

Objektmanagement: Das StP Object Management System (OMS) implementiert das Informationsmodell (*Persistent Data Model*), nach dem die werkzeugunabhängige Ablage der Modellierungsdaten im Repository erfolgt. Außerdem stellt es den Zugriff auf das StP Repository durch eine Menge von Funktionen zur Verfügung. Somit definiert das OMS die Schnittstelle zwischen Werkzeugen der StP Umgebung und dem Repository. Eine werkzeugunabhängige Interaktion zwischen einem Anwendungsentwickler und dem OMS kann mit Hilfe der im Kapitel 4 dargestellten StP Query and Reporting Language oder dem StP Application Program Interface erfolgen.

Repositoryverwaltung: Die Verwaltung des Repositorys erfolgt auf der Grunglage des relationalen Datenbankverwaltungssystems Sybase in einer heterogenen *Client-Server* Umgebung. Der Zugriff auf die Datenbankverwaltung wird direkt vom OMS umgesetzt, ohne dabei einen näheren Einblick in das zugrundeliegende relationale Datenbankmodell zu gewähren. Mit Hilfe von Sybase SQL oder ähnlicher StP unabhängiger Datenbankwerkzeuge kann der Zugriff auf die Repositoryobjekte auch unter Umgehung des StP Object Management Systems realisiert werden. Diese Möglichkeit des Zugriffs auf die Entwicklungsdatenbank ist jedoch kein Untersuchungspunkt dieser Arbeit.

Gültigkeitsprüfung: StP Umgebungen bieten im Zusammenhang mit der Repositoryverwaltung noch einen weiteren Dienst zur Erzielung von Konsistenz innerhalb des Entwicklungsprozesses. Beim Speichern erstellter Modellierungsobjekte im Repository werden diese auf semantische und syntaktische Korrektheit überprüft. Sind Integritätsbedingungen im Rahmen des *Persistent Data Models* mißachtet oder ist eine Abbildung eines Anwendungsdatentyps auf einen PDM-Datentyp nicht möglich, so kann das betreffende Objekt nicht im Repository abgelegt werden.

2.4 Architekturmerkmale

Zur Unterstützung der Anwendungsentwicklung bieten alle Produkte der StP Familie, wie in der Abbildung 2.2 angedeutet, eine Menge grafischer Werkzeuge, welche die Entwicklungskonzepte in allen Phasen des Softwarelebenszyklus unterstützen [Klees, Schmauch 94].

Diese speziellen grafischen Werkzeuge sind wie sonstige allgemeine oberhalb eines methodenunabhängigen Kerns, bestehend aus dem StP Object Management System, Systemdateien und dem StP Repository, und unterhalb des StP Desktops angeordnet.

Die Abbildung 2.2 zeigt den schematischen Aufbau einer StP CASE-Umgebung am Beispiel von StP/OMT, wobei die weiß unterlegten Komponenten bei einer anderen CASE-Umgebung der StP Produktfamilie durch methodenadäquate Komponenten auszutauschen sind. Die grau unterlegten Grundkomponenten der StP Umgebung stellen die Voraussetzung für die Integration der StP Werkzeuge dar.

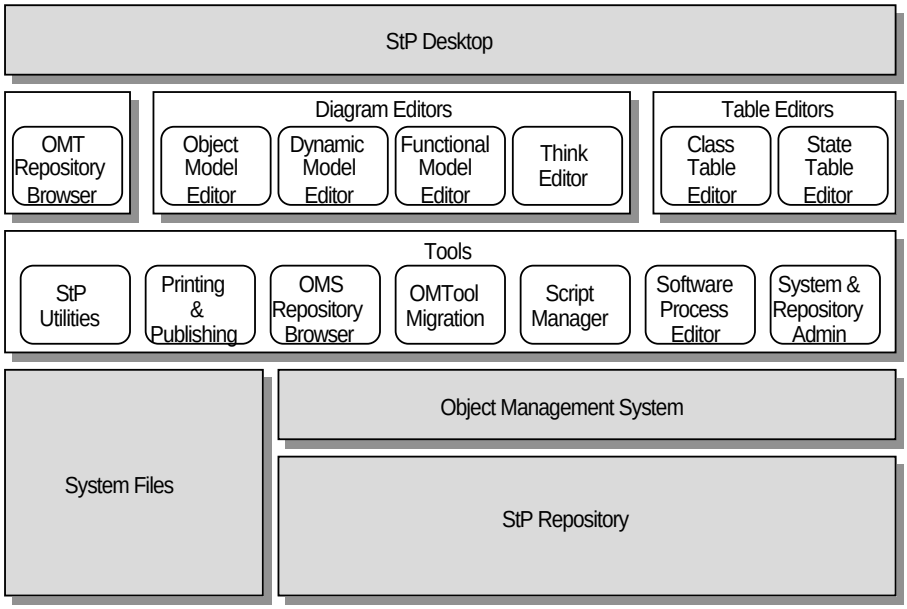

Abbildung 2.2: StP/OMT Architektur

Die Anbindung des Tycoon-Systems erfolgt unterhalb des StP Desktops und oberhalb des StP Kerns, also parallel zu den anderen installierten C ASE-Werkzeugen.

3. Das StP Repository

Jedes einzelne in der Abbildung 2.2 dargestellte StP CASE-Werkzeug benötigt oder erzeugt eine Vielzahl von Informationen innerhalb des Entwicklungsprozesses. In der verteilten *Client-Server* Umgebung entsteht somit innerhalb eines jeden Werkzeugs eine hohe Anzahl von Zugriffsoperationen auf die Datenbasis aller Entwicklungswerkzeuge. Das StP Repository integriert sämtliche Datenbasen aller Werkzeuge der StP Umgebung. Es ist der „zentrale Speicher" der Projektinformation [IDEStPOMS 94].

Um eine anwendungsunabhängige Speicherung der Modellierungsobjekte in der Entwicklungsdatenbank zu erreichen und ein hohes Maß an Konsistenz innerhalb des Anwendungsentwicklungsprozesses zu erzielen, verwendet die StP Umgebung das StP Object Management System. Es beinhaltet neben den im Abschnitt 3.1 dargestellten Aspekten die in der StP Query and Reporting Language enthaltene OMS Query Language und das StP Application Program Interface (API), um auf Repositoryobjekte werkzeugunabhängig zugreifen zu können.

3.1 Objektmanagement

Das Objektmanagement ist innerhalb des OMS durch eine Menge von Funktionen realisiert, welche die Schnittstelle zwischen Anwendungsprogrammen und dem StP Repository definieren. Die Ablage der Projektinformation erfolgt anwendungsunabhängig auf der Grundlage des *Persistent Data Model* (PDM).

3.1.1 Persistent Data Modell (PDM)

Das *Persistent Data Model* ist ein konzeptuelles Schema, welches die Attribute und Beziehungen von Repositoryobjekten auf der Basis abstrakter Datentypen definiert. Somit entsteht für jede Anwendung die gleiche Zugriffssicht auf die im Repository abgelegten Objekte [IDEStPOMS 94].

Die in der Abbildung 3.1 dargestellten abstrakten Datentypen stehen in einer exklusiven Subtypbeziehung. Das heißt, daß jeder Typ in einer Sub-/Supertypbeziehung zu einem anderen Typ steht. Der *CASE Type* ist der Supertyp aller anderen Typen dieses Datenmodells. Supertypen, wie der *CASE Type*, der *Annotated Object Type*, der *Reference Type* und

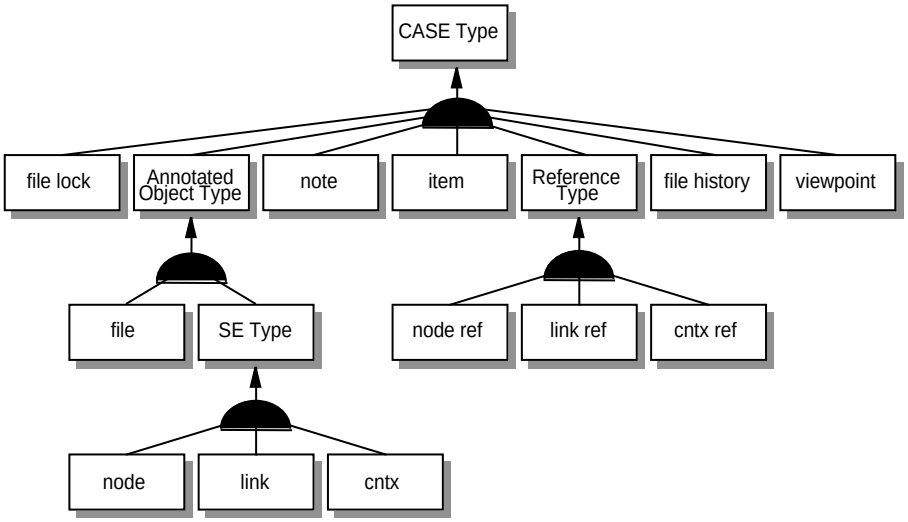

Abbildung 3.1: StP Persistent Data Modell (PDM)

der *Software Engineering (SE) Type* werden als „Kategorien“ bezeichnet und können nicht direkt im Repository gespeichert werden. Nur die Subtypen, die nicht Supertypen anderer Typen sind, können als Repositoryobjekte instantiiert werden. Eine Interpretation der PDM-Typen erfolgt an dieser Stelle nicht. Tabellen aller PDM-Typen und ihrer Attribute befinden sich in [IDEStPOMS 94].

3.1.2 PDM-Typen und Beziehungen

Abgesehen von den konzeptuellen Sub-/Supertypbeziehungen stehen die PDM-Typen noch in einer, für die Modellierung der Tycoon-Bibliothek wesentlichen Beziehung: Für die Ablage der Modellierungsobjekte im Repository gelten referentielle Integritätsbedingungen [IDEStPOMS 94].

Die in der Abbildung 3.2 dargestellten Integritätsbedingungen zur Speicherung von Objekten im Repository haben einen großen Einfluß auf die im Abschnitt 5.4 vorgeschlagene Modularisierung der zu erstellenden Bibliothek auf der Tycoon-Ebene hinsichtlich ihrer Typisierung.

Beispielsweise kann ein `cntx` Objekt nicht im Repository abgelegt werden, solange das referenzierte `link` Objekt dort noch nicht vorhanden ist. In der Abbildung 3.2 zeigt ein Pfeil von einem abhängigen Objekttypen auf einem unabhängigen, dessen Existenz die Voraussetzung für die Ablage des von diesem Objekttypen abhängigen Objekts im Repository darstellt.

Objektattribute, deren Name auf `_id` endet, verweisen also auf ein unabhängiges Repositoryobjekt. Ein mit einem Stern (`*`) gekennzeichnetes Attribut eines Repositoryobjekts, zum

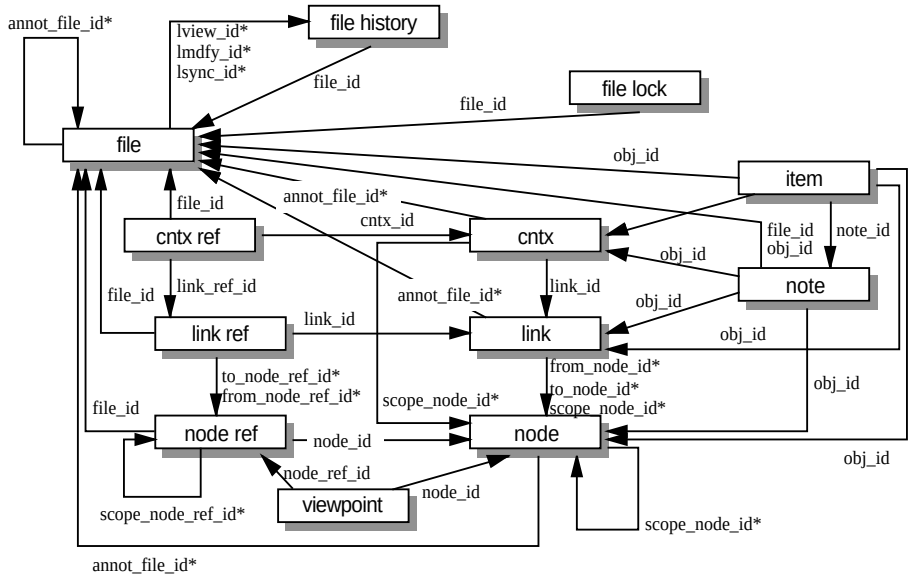

Abbildung 3.2: Integritätsbedingungen zwischen PDM-Typen

Beispiel `annot_file_id*`, kann den Wert „0“ annehmen, was bedeutet, daß das Object Management System bei der Speicherung dieses Objekts im Repository nicht auf der Existenz des von diesem Objekt unabhängigen Objekttyps besteht.

3.2 Anwendungssicht auf das StP Repository

Die Verbindung zwischen den Datentypen von Anwendunsprogrammen und denen des *Persistent Data Models* wird mit Hilfe der „Anwendungstypen“-Datei `app.types` realisiert. Die Anwendungssicht auf das StP Repository ist in der Abbildung 3.3 dargestellt [IDEStPOMS 94].

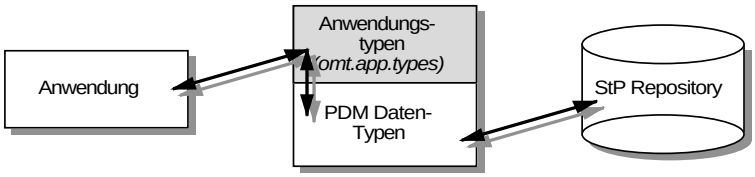

Abbildung 3.3: Anwendungssicht auf das StP Repository

In der momentanen DBIS-Installation von Software through Pictures trägt die Abbildungsdatei die Bezeichnung `omt.app.types`. Hierbei handelt es sich um eine um beliebige Anwen-

dungsdatentypen erweiterbare Datei, welche die von Anwendungsprogrammen referenzierten Anwendungsdatentypen auf die vom StP Repository verwendeten PDM-Datentypen abbildet [IDEStPOMS 94].

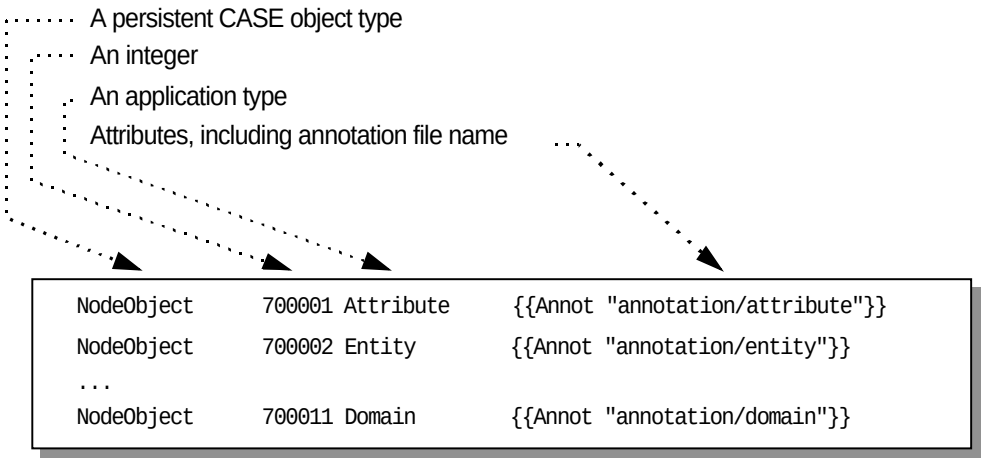

Abbildung 3.4: Abbildung von Anwendungsdatentypen auf PDM-Datentypen

In der Abbildung 3.4 ist ein Ausschnitt der erweiterbaren Datei `app.types` wiedergegeben, in dem einige Anwendungsdatentypen definiert sind (*Application Types*), die auf einen *Persistent CASE Object Type* (hier: `NodeObject`) abgebildet werden. Dabei wird jedem Anwendungsdatentypen ein eindeutiger Wert vom Typ `int` zugeordnet. Jeder *Persistent CASE Object Type* ist ein gültiger PDM-Datentyp (hier: `node`), dessen Attribute an jeden, auf diese Weise definierten Anwendungsdatentypen, vererbt werden.

Zusätzlich zu den geerbten Eigenschaften kann ein Anwendungsdatentyp um zusätzliche Attribute spezialisiert werden, was mit Hilfe einer sogenannten *Annotation* Schablone geschieht (*Annotation File*). Hierbei handelt es sich jeweils um eine, die neuen Attribute spezifizierende, Datei.

Eine tiefergehende Erläuterung der Abbildung jedes einzelnen Anwendungsdatentypen dieser Datei soll an dieser Stelle nicht erfolgen.

4. Zugriffsmöglichkeiten auf StP Repositoryobjekte

Das StP Object Management System stellt dem Anwender neben dem StP Repository Browser die werkzeugunabhängigen Zugriffsmöglichkeiten der StP Query and Reporting Language (QRL) und des StP Application Program Interface (API) zur Verfügung, um direkt auf Repositoryobjekte zugreifen zu können.

Im Rahmen der StP QRL müssen vom Anwender QRL Skripte erstellt werden, in welchen der Repositoryzugriff mit Hilfe der OMS Query Language realisiert wird. Dahingegen werden die StP API Funktionen in C-Programme integriert und zur Laufzeit des C-Programms ausgeführt.

4.1 StP Query and Reporting Language Skripte

StP QRL Skripte sind als integraler Bestandteil des StP Query and Reporting Systems (QRS) zu betrachten, welches dem Anwender ermöglicht, werkzeugunabhängig unter Verwendung von QRL Skripten auf Repositoryobjekte zuzugreifen. Dabei werden mit Hilfe des StP Query and Reporting Processors (qrp) die in einem QRL Skript enthaltenen QRL Befehle im Rahmen der Skriptinterpretation in OMS Anfragen umgesetzt, um so den eigentlichen Zugriff auf die Repositoryobjekte über die Schnittstelle des StP Object Management Systems zu erzielen. Eine Darstellung des QRS ist in der Abbildung 4.1 wiedergegeben [IDEStPQRS 94].

Aufgaben des StP Query and Reporting Systems

Die Hauptaufgabe des StP Query and Reporting Systems besteht darin, einen Mechanismus für das Erstellen und das Ausführen von QRL Skripten und das Ausgeben ihrer Resultate bereitzustellen. Konkret leistet das QRS die Skriptinterpretation, die Realisierung des Zugriffs auf Repositoryobjekte und die Ausgabe der Anfrageergebnisse in formatierter oder unformatierter Form.

Komponenten des StP Query and Reporting Systems

Zu den wichtigsten Bestandteilen des Query and Reporting Systems gehören unter anderem der StP Script Manager, die StP Query and Reporting Language (QRL) und der StP Query and Reporting Processor (qrp).

Der Script Manager bildet die grafische Benutzeroberfläche für die Handhabung des QRS. Im Rahmen dieser Arbeit wird aber nicht weiter auf den Script Manager eingegangen, da er im Hinblick auf die spätere Realisierung der StP Bibliothek auf der Tycoon-Ebene nicht von Bedeutung sein wird. Alle Funktionen des Script Managers können auch über die Kommandooberfläche initialisiert und durchgeführt werden. Nähere Informationen über für diese Arbeit relevante Teilaspekte der QRL sind im Abschnitt 4.1.1 dargestellt. Hinweise zur Interpretation von QRL Skripten und qrp Optionen folgen ebenfalls im nächsten Abschnitt.

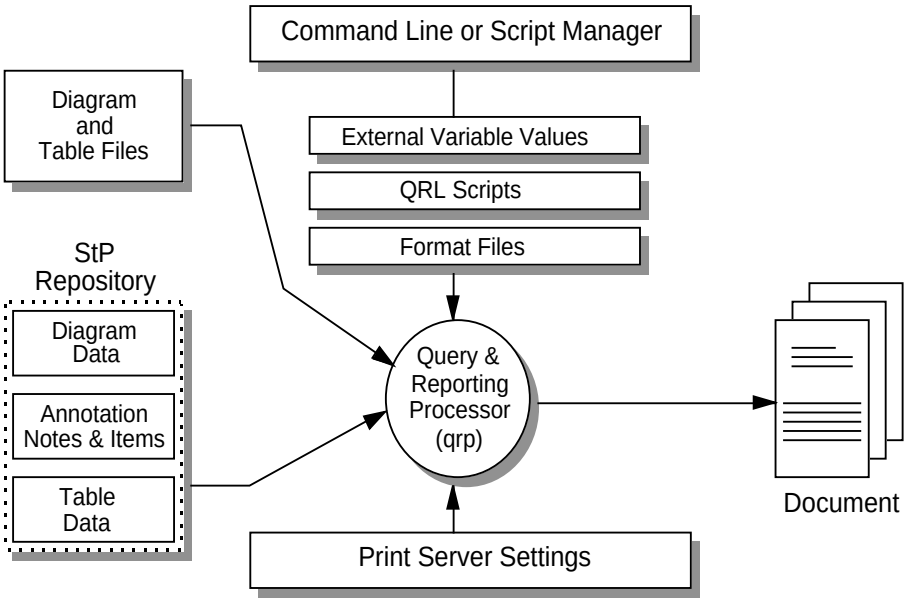

Abbildung 4.1: StP Query and Reporting System (QRS)

4.1.1 StP Query and Reporting Language (QRL)

Die Firma IDE betrachtet ihre Query and Reporting Language als eine *"high-level, structured, C-like script language that includes familiar programming features such as types, parameterized function calls, and control constructs"* [IDEStPQRS 94].

Diese Arbeit befaßt sich nicht mit einer detaillierten Erläuterung der Skriptsprache, sondern ihre Intention liegt darin, einzelne Aspekte soweit zu erklären, wie sie zum weiteren

Verständnis des im Abschnitt 4.1.2 aufgeführten QRL Skripts erforderlich sind.

Aufgabe der StP Query and Reporting Language

Mit Hilfe der StP Query and Reporting Language können „vollwertige“ Anwendungen erstellt werden. Die Betrachtung der Mächtigkeit der QRL bezieht sich hier in erster Linie auf Sprachkonstrukte für die Realisierung des Repositoryzugriffs. Die Skriptsprache enthält für die Entwicklung von StP QRL Applikationen [IDEStPQRS 94]:

- Variablen, Datentypen (primitive Typen, PDM-Typen, abstrakte Typen einschließlich Enumerations-, Listen- und Mengentypen)
- Operatoren (arithmetische, vergleichende, logische und Operatoren für Typkonvertierungen)
- Kontrollfluß (Schleifen und Verzweigungen)
- Funktionen (Typkonvertierung, *Browsing, String Manipulation*, System Ein- und Ausgaben, Enumerations-, Listen- und Mengenoperationen und OMS Schnittstellenoperationen)

Um den in der Abbildung 4.2 dargestellten Zugriff auf Repositoryobjekte zu konkretisieren, werden im folgenden die OMS Schnittstellenfunktionen hauptsächlich betrachtet. In diesem Zusammenhang spielen die PDM-Typen, siehe Abschnitt 3.1.1, eine gewichtige Rolle.

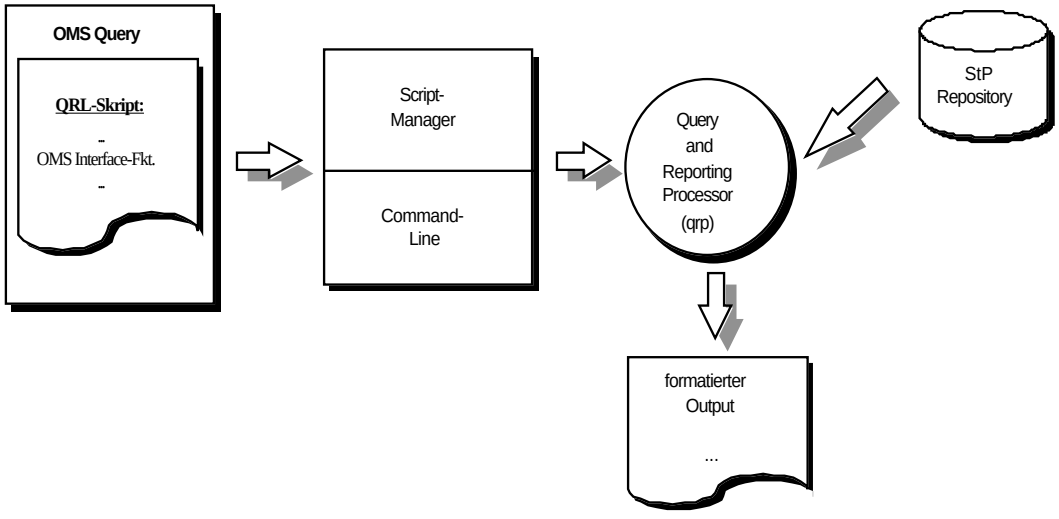

Abbildung 4.2: StP Repository Zugriff

StP QRL Skripte

Das Erstellen von StP QRL Skripten ist auf zwei Arten zu realisieren: Entweder benutzt man den Script Manager oder einen konventionellen Texteditor zum Anlegen oder Editieren von Skriptdateien. Das Ausführen der angelegten Dateien wird im Rahmen der Interpretation von StP QRL Skripten behandelt. Prinzipiell müssen die folgenden Komponenten innerhalb eines QRL Skripts enthalten sein:

```
void
main ()
{
   print_line("Ausfuehrungsteil des Skripts");
}
```

- Obligatorisch muß jedes QRL Skript eine `main`-Funktion enthalten, welche anzeigt, an welcher Stelle der Ausführungsteil des Skripts beginnt.
- Ebenfalls obligatorisch ist der Rückgabetyp der `main`-Funktion, in diesem Fall `void`.
- Von geschweiften Klammern eingeschlossen folgt dann der eigentliche Ausführungsteil des Skripts, jeder einzelne QRL Befehl wird durch ein Semikolon abgeschlossen.
- Verschachtelungen innerhalb der QRL Befehle sind auf der Basis der StP QRL Syntax möglich.

Auf der Kommandooberfläche wird ein QRL Skript mit dem folgenden Befehl ausgeführt bzw. zur Interpretation aufgerufen:

```
qrp script_name [-o output_file] [-x ext_var_name ext_var_value]
                [-f formats_file] [-t target] [-I include_dir]
```

In der Tabelle 4.1 sind alle möglichen qrp Optionen wiedergegeben [IDEStPQRS 94].

StP PDM-Typen

Die Grundlage für das Ablegen von Objekten im StP Repository bildet das im Abschnitt 3.1.1 dargestellte *Persistent Data Model* (PDM). Die folgenden PDM-Typen werden von der StP Query and Reporting Language zur Verfügung gestellt [IDEStPOMS 94]: `cntx`, `cntx_ref`, `file`, `file_hist`, `file_lock`, `item`, `link`, `link_ref`, `node`, `node_ref`, `note`, `viewpoint`.

Alle Zugriffsoperationen auf Objekte innerhalb des Repositorys basieren auf diesen abstrakten Datentypen. Der Zugriff selbst erfolgt mit Hilfe der OMS Schnittstellenfunktionen. Eine

Option	Argument	Beschreibung
	<script_name>	auszuführendes Skript
-o	<output_file>	Datei, in welche die Ausgabe erfolgen soll, (stdout) ist *default*.
-x	<ext_var_name> <ext_var_value>	Spezifiziert einen im Skript verwendeten externen Variablennamen und -wert.
-f	<formats_file>	Spezifiziert die Datei, die die Information zur Formatierung der Ausgabe enthält.
-t	ascii|mif|leaf	Spezifiziert das Format der Ausgabe, ASCII (ascii) ist *default*, FrameMaker (mif) oder Interleaf (leaf).
-I	<include_dir>	Spezifiziert andere als in ToolInfo-Datei angegebene *Include*-Pfade.

Tabelle 4.1: qrp Optionen

Interpretation der hier genannten Typen erfolgt an dieser Stelle nicht, da diese keine Voraussetzung für die Umsetzung des eigentlichen Repositoryzugriffs darstellt.

StP OMS Schnittstellenfunktionen

Die StP Query and Reporting Language stellt eine Reihe von Funktionen zur Verfügung, die eine Schnittstelle zwischen dem Anwender und dem StP Object Management System definieren und somit, innerhalb eines QRL Skripts aufgerufen, Information über Repositoryobjekte zurückgeben. Eine Auswahl der entsprechenden Funktionen ist in der Tabelle 4.2 aufgeführt [IDEStPQRS 94].

4.1.2 Das StP QRL Skript Select.qrl

Um den Zugriff auf Repositoryobjekte mit Hilfe eines StP QRL Skripts exemplarisch darzustellen, wurde das im Anhang A.1.1 wiedergegebene Skript `Select.qrl` entwickelt. Dabei wird auf komplizierte Schleifen und Verzweigungen innerhalb des Ablaufs zugunsten der Übersichtlichkeit verzichtet. Die im vorigen Abschnitt aufgeführten OMS Schnittstellenfunktionen stehen im Vordergrund der Betrachtung und werden exemplarisch ausgeführt.

Skriptinhalt

Das QRL Skript `Select.qrl` wurde mit einem konventionellen Texteditor erstellt und ohne qrp Optionen interpretiert. Eine spezielle Formatierung der Skriptausgabe erfolgte nicht (*Default*: ascii), allerdings wurde das Skriptresultat nachträglich in die Datei `Select_Output.txt` übernommen und zwecks besserer Übersichtlichkeit aufbereitet (siehe

Funktion	Rückgabetyp	Beschreibung
for_each_in_select(string oms_query,<PDM Typ> query_obj)	<PDM Typ>	Zugriff auf Repository-objekte
find_by_query(string oms_query)	<PDM Typ>	Rückgabe des ersten, gefundenen Objekts der Anfrage
selection_count (string oms_query)	int	Rückgabe der Anzahl gefundener Anfrageobjekte
id_list_create(string oms_query, string id_list_name)	void	Erstellen einer ID-Liste der ausgewählten Objekte
id_list_free(string id_list_name)	void	Löschen der angegebenen ID-Liste
list_select(string oms_query)	list	Führt OMS Anfrage aus und gibt Liste mit dem Resultat zurück.

Tabelle 4.2: OMS Schnittstellenfunktionen

Anhang A.1.2). Auf der Kommandooberfläche erfolgte der Aufruf zur Skriptinterpretation in der folgenden Weise:

```
qrp Select.qrl
```

In diesem Abschnitt sind nur einige Ausschnitte des gesamten QRL Skripts aufgeführt. Die obligatorischen Skriptelemente sind zwecks besserer Übersichtlichkeit vernachlässigt worden.

Variablendeklaration und -initialisierung:

```
cntx_ref cntx_ref_var;
cntx cntx_var;
link link_var;
node node_var;
int zaehler;
int id = 14209;
string query = "node[type == OMTEvent && annot_file_id == 0]";
```

QRL Befehle für den Zugriff und die Ausgabe einiger im StP Repository abgelegter Werte:

OMS Anfrage Nr. 1:

```
for_each_in_select("cntx_ref", cntx_ref_var)

print_line("cntx_ref_id: \"" + cntx_ref_var.id + "\" cntx_id: "
+ cntx_ref_var.cntx_id);
```

Diese OMS Anfrage sucht nach `cntx_ref` Objekten im Repository. Anschließend werden die Werte der Attribute `id` und `cntx_id` ausgegeben:

```
cntx_ref_id: "54602" cntx_id: 54601
cntx_ref_id: "54604" cntx_id: 54603
```

OMS Anfrage Nr. 2:

```
for_each_in_select("link[id >= 54400]", link_var)

print_line("link_id: \"" + link_var.id
+ "\" from_node_id: " + link_var.from_node_id
+ "\" to_node_id: " + link_var.to_node_id);
```

Diese OMS Anfrage sucht nach `link` Objekten im Repository, deren Attribut `id` die angegebene Bedingung erfüllt. Anschließend werden die Werte der Attribute `id`, `from_node_id` und `to_node_id` ausgegeben:

```
link_id: "54403" from_node_id: 402" to_node_id: 301
link_id: "59202" from_node_id: 4001" to_node_id: 401
link_id: "59203" from_node_id: 401" to_node_id: 4001
```

OMS Anfrage Nr. 3:

```
print(find_by_query("node[id == "+id+"]"));
```

Diese OMS Anfrage sucht nach `node` Objekten im Repository, deren Attribut `id` die angegebene Bedingung erfüllt. Anschließend werden alle Attribute und deren Werte ausgegeben:

```
{
id = 14209
name = send message request
type = OMTEvent
scope_node_id = 0
sig =
annot_file_id = 0
}
```

OMS Anfrage Nr. 4:

```
zaehler = selection_count(query);
```

```
print("Anzahl gefundener Objekte, die vom Typ OMTEvent sind"
+ " und den Wert 0 als annot_file_id besitzen: " + zaehler);
```

Diese OMS Anfrage sucht nach den in der Anfrage spezifizierten Repositoryobjekten. Anschließend wird deren Anzahl ausgegeben:

```
Anzahl gefundener Objekte, die vom Typ OMTEvent sind und
den Wert 0 als annot_file_id besitzen: 41
```

Der Zugriff auf die Attributwerte der PDM-Typen erfolgt nach der Zuweisung an die entsprechende Variable mit Hilfe der Punktnotation. Alle PDM-Typen einschließlich ihrer Attribute und Relationen sind in [IDEStPOMS 94] dargestellt.

Die vollständige Ausgabe des gesamten Skripts `Select.qrl` befindet sich in der Datei `Select_Output.txt` im Anhang A.1.2.

4.2 C-Programme auf der Basis des StP Application Program Interface (API)

Neben der Einbettung von OMS Anfragen in Query and Reporting Language Skripte ist es ebenfalls möglich, die Repository-Anfrageanweisungen in Funktionen des StP Application Program Interface in der Form von an die API Funktionen übergebenen Argumenten (als `Strings`) zu integrieren. Desweiteren bietet das StP API noch andere Funktionen für den direkten Zugriff auf Repositoryobjekte.

4.2.1 StP Application Program Interface

Das StP Application Program Interface bildet eine Schnittstelle zwischen C-Anwendungsprogrammen und dem StP Repository. Es stellt dem Anwender eine Reihe von in C-Programme integrierbare Funktionen und Datentypen zur Verfügung, um auf Repositoryobjekte zugreifen bzw. das Repository selbst verwalten zu können. Zu den für diese Arbeit relevanten Aufgaben der API Funktionen gehören [IDEStPOMS 94]:

- Erstellen, Auffinden und Löschen von Repositoryobjekten
- Verwalten des Hauptspeichers und des Repositorys
- Interpretieren von OMS Anfragen
- Manipulieren von Attributwerten der Repositoryobjekte
- Durchführen von StP Repositorytransaktionen

In der Tabelle 4.3 sind exemplarisch einige wichtige StP API Funktionen zum Erfüllen der oben genannten Aufgaben wiedergegeben. Weitere Funktionen sind in [IDEStPOMS 94] aufgeführt.

StP API Funktionen und Typen

Tabelle 4.3 stellt die im Abschnitt 4.2.2 verwendeten Funktionen dar. Die konkrete Typisierung der API Funktionen ist für alle Datentypen des *Persistent Data Models* in [IDEStPOMS 94] wiedergegeben. Eine Interpretation der im Abschnitt 4.2.2 verwendeten Datentypen erfolgt an dieser Stelle nicht.

Außer den erwähnten und im Abschnitt 4.2.2 benutzten Funktionen stellt das StP API auch noch weitere Funktionen im Rahmen von *Collection-*, *Big Collection-*, Transaktionstypen und ID-Listen bereit [IDEStPOMS 94].

Bei diesen Typen handelt es sich um „Hilfstypen“, welche nicht im Repository abgelegt werden können. Der *Collection* Typ wird benötigt, um eine Mehrzahl von Objekten in eine einzelne Einheit zusammenzufassen. Sollte eine OMS Anfrage ein Resultat liefern, bei dem die Anzahl gefundener Repositoryobjekte zu groß ist, um sie in den Hauptspeicher zu laden, wird der *Big Collection* Typ verwendet. Der Transaktionstyp stellt dem Anwendungsentwickler einen Mechanismus zur Verfügung, mit dessen Hilfe eine feste Anzahl von Repositoryoperationen kontrolliert werden kann. ID-Listen sind kurzfristige „Aufbewahrungsorte“ für das Resultat einer OMS Anfrage, um es als Anfrageargument weiter benutzen zu können.

Desweiteren ist eine individuelle Fehlerbehandlung realisiert. Standardmäßig gibt das OMS beim Auftreten eines Fehlers eine Fehlernummer und eine Fehlermeldung aus. Innerhalb des API existiert eine Funktion `oms_set_oms_err_handler`, mit deren Hilfe Standardfehlermeldungen abgefangen werden können.

Systemvoraussetzungen

Um auf die einzelnen Funktionen des StP Application Program Interface auch tatsächlich zugreifen zu können, ist es erforderlich, gewisse Systemvoraussetzungen zu beachten:

1. Innerhalb des C-Anwendungsprogramms müssen die einzubeziehenden *Header*-Dateien eingebunden werden:

   ```
   #include "oms_stdinc.h",
   #include <stdio.h>
   ```

2. Aufgrund einiger Fehlermeldungen müssen in der Datei `libtools.h` im Verzeichnis `.../StP/templates/ct/include/` die Zeilen 13, 206, 499, 500, 501, 504, 510, 514, 515, 516 auskommentiert werden. Eine nachteilige Wirkung auf die Funktionalität des im Abschnitt 4.2.2 dargestellten C-Programms kann nicht festgestellt werden. Welche

Funktion	Beschreibung
oms_<pdm_typ>_create	Erzeugt ein neues Repositoryobjekt im Hauptspeicher und gibt einen darauf weisenden Zeiger zurück.
oms_<pdm_typ>_find	Sucht im aktuellen Repository nach dem mit den Funktionsparametern spezifizierten Objekt. Dieses wird in eine neue oms_<pdm_typ>_tp Struktur kopiert. Ein darauf weisender Zeiger wird zurückgegeben. Ansonsten wird der NULL-Zeiger zurückgegeben.
oms_<pdm_typ>_find_by_id	Sucht im aktuellen Repository nach dem mit den Funktionsparametern spezifizierten Objekt. Dieses wird in eine neue oms_<pdm_typ>_tp Struktur kopiert. Ein darauf weisender Zeiger wird zurückgegeben. Ansonsten wird der NULL-Zeiger zurückgegeben.
oms_<pdm_typ>_find_by_query	Sucht im aktuellen Repository nach dem mit den „OMS Anfragebedingungen“ übereinstimmenden Objekt. Dieses wird in eine oms_<pdm_typ>_tp Struktur kopiert. Ein darauf weisender Zeiger wird zurückgegeben. Ansonsten wird der NULL-Zeiger zurückgegeben.
oms_<pdm_typ>_free	Nimmt die Adresse des übergebenen Zeigers auf die oms_<pdm_typ>_tp Struktur und gibt den damit assoziierten Hauptspeicherbereich frei.
oms_<pdm_typ>_print	Nimmt die Adresse des übergebenen Zeigers auf die oms_<pdm_typ>_tp Struktur und gibt alle Attributnamen und deren Werte in einer zu spezifizierenden Datei aus.
oms_<pdm_typ>_ <attribut>_asgn	Weist einem Repositoryobjekt-Attribut einen neuen Wert zu.
oms_<pdm_typ>_update	Aktualisiert oder fügt im Hauptspeicher angelegte Strukturen im StP Repository als Objekte ein. Tritt ein Fehler auf oder sind referentielle Integritätsbedingungen nicht erfüllt, gibt die Funktion OMS_FAIL zurück, ansonsten OMS_SUCCEED.
oms_sys_repos_open	Öffnet das spezifizierte Repository für den Zugriff.
oms_repos_close	Schließt das aktuelle Repository nach dem Abschluß der Zugriffsoperationen.

Tabelle 4.3: Ausgewählte StP Application Program Interface Funktionen

Konsequenzen sich aber für andere Anwendungsprogramme, welche API Funktionen aufrufen, ergeben, kann nicht vorhergesagt werden.

3. Der Repositoryzugriff erfordert das Vorhandensein einiger Systemumgebungsvariablen. Im einzelnen handelt es sich um die Variablen (mit „..." beginnende Angaben bezeichnen relative Pfadangaben):

```
LD_LIBRARY_PATH=.../StP/libsparc:/usr/openwin/lib:/opt/sv/lib
PATH=.../StP/binsparc:.../StP/sybase_10.0.1/sparc/bin:
     /usr/ccs/bin:/opt/SUNWspro/SW3.0.1/bin
SHLVL=3
IDE_PRODUCT=omt
ToolInfo=.../StP/ToolInfo/IDE
SYBASE=.../StP/sybase_10.0.1/sparc
DSQUERY=STP_SERVER
SYS_TEN=true
STP_SERVER_PASSWD=welcome
```

Sind diese Systemvoraussetzungen erfüllt, kann mit den im Abschnitt 4.2.2 erläuterten Schritten zur Programmausführung fortgesetzt werden. Fehlermeldungen aufgrund nicht vorhandener oder auffindbarer Systemkomponenten sollten nicht auftreten.

4.2.2 Das C-Programm api3.c

Um den Zugriff auf Repositoryobjekte mit Hilfe von in C-Programmen integrierten StP Application Program Interface Funktionen exemplarisch darzustellen, wurde das im Anhang A.2.1 wiedergegebene C-Programm `api3.c` entwickelt. Dabei wurde auf komplizierte Schleifen und Verzweigungen innerhalb des Aufbaus zugunsten der Übersichtlichkeit verzichtet. Die im Abschnitt 4.2.1 aufgeführten API Funktionen stehen im Vordergrund der Betrachtung und werden exemplarisch ausgeführt.

Vorgehen zur Programmausführung

Sind die im Abschnitt 4.2.1 beschriebenen Systemvoraussetzungen für die Ausführung der StP Application Program Funktionsaufrufe realisiert, kann innerhalb der DBIS StP Umgebung wie folgt vorgegangen werden, um auf StP Repositoryobjekte zuzugreifen:

1. Erstellen des C-Quellprogramms, hier: `api3.c`.
2. Übersetzen des C-Quellprogramms:

   ```
   cc -c api3.c -I .../StP/templates/IDE/include
   ```

3. Verbinden des Objektkodes der Datei `api3.o` mit dem Kode der OMS Objektdateien:

```
cc api3.o -o api3
-L .../StP/libsparc -L .../StP/sybase_10.0.1/sparc/lib
-L /usr/lib -lideoms -lideparse -lideeditor -lideoms -lidetools
-lsybdb -lm -lidestub -lnsl
```

4. Aufruf des ausführbaren Programms, hier: `api3`.

Programminhalt und -interpretation

Das C-Programm `api3.c` hat zusammengefaßt die Aufgabe, ein Repositoryobjekt mit den im Deklarationsteil initialisierten Attributwerten zu generieren, mehrmals auf verschiedene Arten nach diesem Objekt im entsprechenden StP Repository zu suchen und schließlich das Repository zu aktualisieren. Zu diesem Zweck werden die im Abschnitt 4.2.1 erläuterten API Funktionen verwendet. Das Programm `api3.c` wurde zweimal hintereinander aufgerufen: Beim ersten Mal war das entsprechende Objekt schon im Repository vorhanden, beim zweiten Mal sollte ein neues Objekt generiert werden.

In diesem Abschnitt sind nur einige wesentliche Ausschnitte des gesamten Programms `api3.c` wiedergegeben, die zum Verständnis der Vorgehensweise bei API Funktionsaufrufen erforderlich sind. Desweiteren wird an dieser Stelle nur die Initialisierung der Attributwerte des ersten Programmlaufs dargestellt.

Variablendeklaration und -initialisierung (erster Programmaufruf):

```
char *name1, *name2, *name3;
char *name = "ausgedachter Node";
oms_app_type_tp type = 701576;
oms_object_id_tp scope = 0;
char *sig = "hallo";
oms_object_id_tp annot = 5;
FILE *fp;

oms_node_tp *node1, *node2, *node3, *node4;
char buf[1024];
oms_status_tp status;
```

Die hier initialisierte Struktur entspricht einem PDM-Objekt `node` mit dem Namen `ausgedachter Node` vom Typ `OMTEvent`, mit der `scope_node_id = 0`, der Signatur `hallo` und der `annot_file_id = 5`.

API Anweisungen für den Repositoryzugriff:

Spezifikation des zu öffnenden Repositorys:

```
oms_sys_repos_open("../StP/project", "tutor");
```

Der StP Repositoryzugriff erfolgt auf das System `tutor` innerhalb des angegebenen Verzeichnisses.

Generierung einer Repositoryobjekt äquivalenten Struktur im Hauptspeicher:

```
node1 = oms_node_create(name, type, scope, sig);
```

Der Zeiger `node1` weist auf die erzeugte `node` Struktur im Hauptspeicher, welche die übergebenen Attributwerte enthält.

Auffinden des spezifizierten `node` Objekts im StP Repository:

```
node2 = oms_node_find(name, type, scope, sig);
```

Der Zeiger `node2` weist auf die erzeugte `node` Struktur im Hauptspeicher, nachdem das spezifizierte Objekt im Repository aufgefunden wurde. Ein weiterer Zeiger `node3` weist später genau wie `node4` auf eine Kopie dieser Struktur.

Auffinden des `node` Objekts im StP Repository mit Hilfe einer OMS Anfrage:

```
sprintf(buf, "node[name='%s' && type=%d && scope_node_id=%ld &&
sig='%s']", name, type, scope, sig);
node4 = oms_node_find_by_query(buf);
```

OMS Query and Reporting Language Anweisungen können als `String` an die API Funktion `oms_node_find_by_query` übergeben werden. Zu beachten ist dabei, daß die OMS Anfrage das Repository abfragt und nicht den Hauptspeicher. Einzelheiten über OMS Anfragen auf der Basis von StP Query and Reporting Language Anweisungen sind in [IDEStPQRS 94] enthalten.

Ausgabe aller Attributnamen und -werte:

```
oms_node_print(node1, fp);
```

Die Attributnamen und -werte der Struktur, auf die der Zeiger `node1` zeigt, werden ausgegeben.

Freigabe des mit einer Struktur assoziierten Hauptspeicherbereichs:

```
oms_node_free(&node4);
```

Der mit der Struktur, auf die der Zeiger `node4` weist, assoziierte Hauptspeicherbereich wird freigegeben.

Aktualisierung des Repositorys:

```
oms_node_update(node1);
```

Die im Hauptspeicher abgelegte Struktur, auf die der Zeiger `node1` weist, wird im StP Repository persistent abgelegt.

Attributwertänderung bei einem persistenten Repositoryobjekt:

```
oms_node_annot_file_id_asgn(node1, 0);
oms_node_update(node1);
```

Dem Strukturattribut `annot_file_id` wird der Wert „0“ zugewiesen. Anschließend wird das StP Repository um den Inhalt der gesamten Struktur, auf die der Zeiger `node1` im Hauptspeicher weist, aktualisiert.

Schließung des aktuellen Repositorys:

```
oms_repos_close();
```

Zum Studium der vollständigen, aufbereiteten Ausgabe der Datei `api3.c` (nach zweimaligem Aufruf) verweise ich auf die Datei `api3_Output.txt` im Anhang A.2.2.

4.3 Bewertung

Die OMS Schnittstellenfunktionen und das StP API bilden die Schnittstelle zwischen dem Anwender und dem StP Object Management System, welches für die Verwaltung des Repositorys zuständig ist.

Der Repositoryzugriff kann im Rahmen des StP Query and Reporting Systems realisiert werden. Hierzu werden die in QRL Skripten enthaltenen OMS Schnittstellen-Funktionsaufrufe in OMS Anfragen umgewandelt und daraufhin vom OMS bearbeitet.

Die OMS Schnittstellenfunktionen können problemlos in StP QRL Skripte integriert werden. Desweiteren stellt die Skriptsprache sämtliche, zur Erstellung von Applikationen benötigten Sprachkonstrukte, zur Verfügung. Somit ist ein Zugriff auf die Repositoryobjekte relativ unkompliziert mit der Hilfe eines StP QRL Skripts zu erreichen. Auf alle im StP Repository gespeicherten Attributwerte aller PDM-Typen kann auf diese Art zugegriffen werden.

Für den Umgang mit QRL Skripten gibt es zwei Möglichkeiten, den Script Manager oder die qrp Umgebung auf der Kommandooberfläche. Zwar ist der Script Manager aufgrund seiner fensterorientierten Darstellungsform angenehmer zu benutzen als der zeilenorientierte qrp Befehl. Dafür verfügt der zeilenorientierte Skriptinterpretationsbefehl qrp aber über eine im Zusammenhang mit den dargestellten Optionen – im Hinblick auf die Aufgabenstellung dieser Arbeit – angemessene Funktionalität.

Wie im Abschnitt 4.2.1 gezeigt, stellen die Application Program Interface Funktionen neben den OMS Schnittstellenfunktionen eine geeignete Schnittstelle zwischen dem Anwender und dem StP Repository dar.

Das StP API stellt dem Anwender in C-Programme integrierbare Funktionen zum Erstellen und Auffinden von Repositoryobjekten, zum Verwalten des Hauptspeichers und des StP Repositorys selbst, zum Manipulieren von Repositoryobjekt-Attributwerten, zum Handhaben von *Collection-*, *Big Collection-*, Transaktionstypen und ID-Listen und zur Fehlerbehandlung bei fehlerhaften Repositoryaktionen, zur Verfügung. Desweiteren ist es dem Anwender möglich, eine OMS Anfrage in `String`-Repräsentation der API Funktion `oms_<pdm_typ>_find_by_query` als Parameter zu übergeben.

Die StP Application Program Interface Funktionen können problemlos aus C-Programmen aufgerufen werden. Allerdings sind dabei die Systemvoraussetzungen, siehe Abschnitt 4.2.1, zu berücksichtigen. Das Übersetzen und Binden der entsprechenden Dateien, wie im Abschnitt 4.2.2 beschrieben, stellt beim Vorhandensein sämtlicher Bibliotheksdateien ebenfalls kein Problem dar. Somit ist der Zugriff auf die Repositoryobjekte relativ einfach mit der Hilfe eines C-Programms und der entsprechenden API Funktionen zu erreichen. Auf alle im StP Repository „gespeicherten“ Attributwerte aller PDM-Typen kann auf diese Art zugegriffen werden.

Verglichen mit dem Repositoryzugriff auf der Basis von StP QRL Skripten stellt das StP API ein wesentlich größeres Maß an Funktionalität für den eigentlichen Zugriff zur Verfügung. Die Mächtigkeit von QRL Skripten ist eindeutig in den Möglichkeiten zur formatierten Ausgabe von StP Repository Anfragen zu begründen. Im Hinblick auf die zu realisierende Anbindung des Tycoon-Systems an die StP Umgebung ist die Benutzung des Application Program Interface der von StP QRL Skripten vorzuziehen.

Wie der Zugriff auf StP Repositorydaten aus dem Tycoon-System heraus im Rahmen von StP API Funktionsaufrufen endgültig realisiert werden kann, wird zu einem späteren Zeitpunkt im Rahmen dieser Arbeit untersucht werden.

5. Anbindung des Tycoon-Systems an die StP Umgebung

Die Anbindung des Tycoon-Systems an die StP Umgebung erscheint dem Benutzer wie in der Abbildung 5.1 dargestellt.

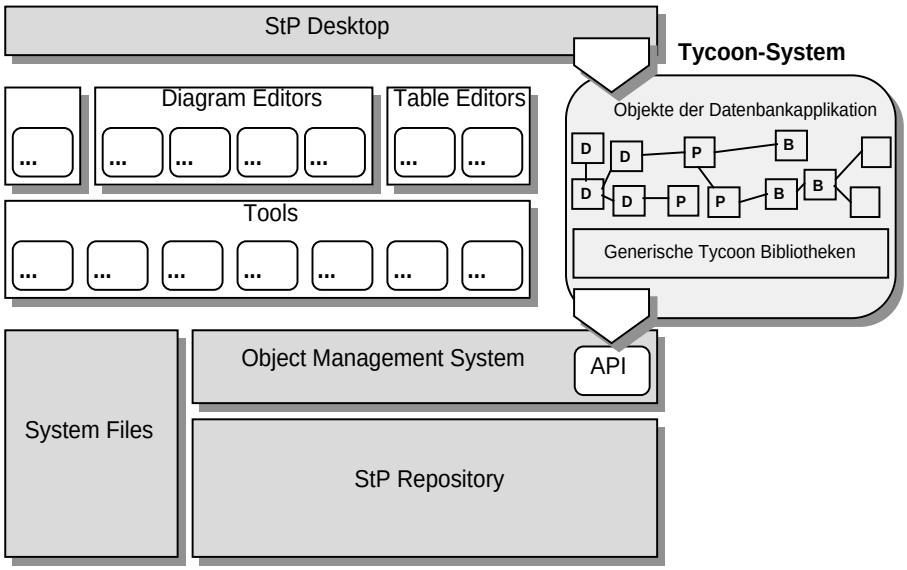

Abbildung 5.1: Anbindung des Tycoon-Systems an die StP Umgebung

Dabei verhält sich das Tycoon-System als integrierte Entwicklungsumgebung hinsichtlich seines Aufrufs und Repositoryzugriffs wie sonstige, in die StP Umgebung integrierte StP Werkzeuge. Der Aufruf des Tycoon-Systems erfolgt über das im Abschnitt 5.1.4 dargestellte Piktogramm innerhalb des StP Desktops, der Zugriff des Tycoon-Systems auf die StP Repositoryobjekte wird über die vom StP Application Program Interface als einem generischen, externen Diensterbringer, zur Verfügung gestellten Funktionen realisiert. Somit kann das Tycoon-System für den Zugriff auf Objekte des StP Repositorys die Funktionalität des

StP Object Management Systems benutzen.

Als Resultat dieser Anbindung des Tycoon-Systems an die StP Umgebung entsteht innerhalb des Tycoon-Systems die generische Bibliothek *stpenv*, die dem Tycoon-Anwender innerhalb des unifizierenden sprachlichen Raums der Tycoon-Umgebung die gesamte, im Abschnitt 5.2 dargestellte, StP Funktionalität für den Repositoryzugriff bietet.

5.1 Das Tycoon-System als integrierte Entwicklungsumgebung

Wird eine StP CASE-Umgebung der Firma Interactive Development Environments gestartet, erscheint der StP Desktop, welcher das Aufrufen von Editoren, des Repository Browsers, des Script Managers, des Print Servers und sonstiger Administrationswerkzeuge ermöglicht.

Der StP Desktop ist an die benötigte Funktionalität der installierten StP Produkte anpaßbar. Sogar StP fremde Befehle, Startskripte oder Programme können in den StP Desktop integriert und von dort aus aufgerufen werden – so auch das Tycoon-System.

5.1.1 Benutzeranpassung des StP Desktop

Das Aussehen und Verhalten des StP Desktops kann auf verschiedene Weise modifiziert werden [IDEStPCust 94]. Zur Vollständigkeit werden hier einige dieser Möglichkeiten erwähnt, ohne daß auf alle von ihnen detailliert eingegangen wird:

- Veränderung der Menüregeln
- Benutzung von `ToolInfo`-Variablen
- Setzen von X-Ressourcen
- Modifikation der Klassenregeln

Meine Vorgehensweise zur Desktopindividualisierung bezieht sich auf die den StP Desktop beschreibenden Klassenregeln und einige ihrer Attribute. Diese Regeln der höchsten Ebene definieren alle vorhandenen Desktopklassen und ihr Verhalten. Jede Klasse muß durch eine Klassenregel spezifiziert werden. In der Abbildung 5.2 sind beispielsweise 26 Klassen im StP Desktop enthalten. Jede aufrufbare Klasse wird dabei durch ein eigenes Piktogramm dargestellt.

Die Spezifizierung der grundlegenden Klassen erfolgt durch Wertzuweisung ihrer Attribute in der Datei `.../StP/templates/IDE/ct/rules/stp.rules`. Die entsprechenden Piktogrammdateien dieser Klassen sind im Verzeichnis `.../StP/templates/IDE/ct/icon` aufzufinden.

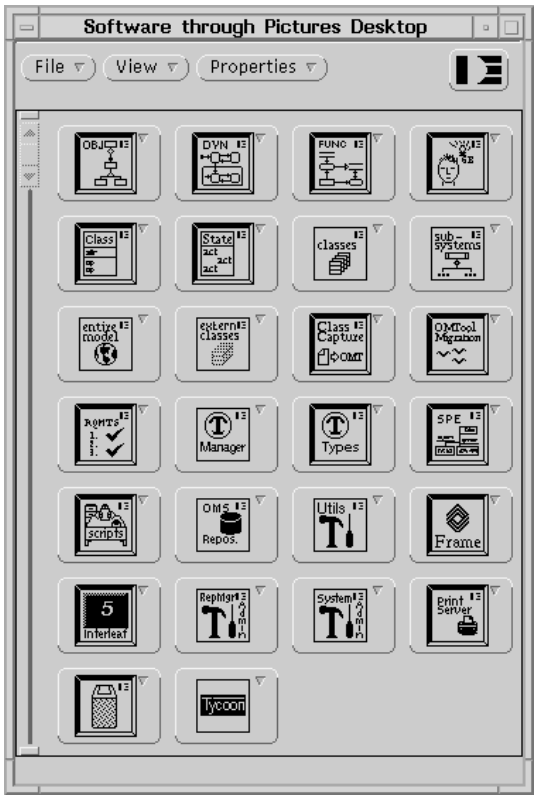

Abbildung 5.2: Software through Pictures Desktop

Modifikationen des StP Desktop können vom Anwender in der dafür vorgesehenen Spezifikationsdatei neuer Klassen vorgenommen werden. Die dafür vorgesehene Datei `user_stp.rules` befindet sich im Verzeichnis `.../StP/templates/IDE/user/ct/rules`.

Die Piktogramme der hinzugefügten Klassen können unter einem beliebigen Pfad hinzugefügt werden. Der Übersichtlichkeit wegen sollten sie aber im Verzeichnis `.../StP/templates/IDE/user/ct/icon` gespeichert werden. Dort befindet sich auch die das Piktogramm des Tycoon-Systemaufrufs spezifizierende Datei `user_tycoon_32.xbm`.

5.1.2 Voraussetzungen der Desktop-Individualisierung

Um den StP Desktop um das Piktogramm der Klasse DBIS-Tycoon zu ergänzen und somit den Aufruf des Tycoon-Systems zu realisieren, sind einige Voraussetzungen zu beachten:

1. Die Datei des in der Abbildung 5.2 dargestellten Tycoon-Piktogramms muß in dem im Abschnitt 5.1.1 wiedergegebenen Piktogrammverzeichnis enthalten bzw. über eine gültige Pfadangabe zugreifbar sein.

2. Die Klasse DBIS-Tycoon ist, wie im Abschnitt 5.1.3 angegeben, in der Datei `user_stp.rules` spezifiziert.

3. Das Tycoon-System ist über das Skript `start <workspace_name>` aufrufbar. Hierbei handelt es sich um ein Startskript, welches einen separaten Tycoon-Prozeß in einem eigenen `cmdtool`-Fenster initiiert. `start` basiert auf dem Startskript `tw <workspace_name>`. Das komplette Skript `start` wird an dieser Stelle nicht detailliert behandelt, ist aber im Anhang A.3 wiedergegeben.

Grundvoraussetzung für den Aufruf von StP/OMT und des Tycoon-Systems sind gültige Zugriffsrechte auf die benötigten Dateien, gültige Pfadangaben und die sonstigen allgemeinen Zugriffsbedingungen (Gruppenzugehörigkeiten etc.) in Mehrbenutzersystemen.

5.1.3 Umsetzung der Desktop-Modifikation

Um die Klasse DBIS-Tycoon dem StP Desktop hinzuzufügen, wurden die folgenden Anweisungen in der im Abschnitt 5.1.1 erwähnten Klassen-Spezifikationsdatei eingefügt:

```
Class Tycoon
{

    { Label "DBIS-Tycoon" }
    { Image "user/ct/icon/user_tycoon_32.xbm" }
    { ShowOnDesktop True }
    { ClassCommand

        { SpaceAfter True }
        { Name "tycoon" }
        { Label "Start Tycoon..." }
        { Command "/local/dbis11/software/StP/scripts/start
        tycoon" }
        { Type Nowin }

    }

}
```

Der Effekt dieser Anweisungen ist der, daß der StP Desktop um das spezifizierte Tycoon-Piktogramm ergänzt wird und ein Aufruf von Tycoon so ermöglicht wird. Eine tiefgreifende Erklärung aller dargestellten Klassenattribute ist in [IDEStPCust 94] enthalten.

5.1.4 Aufruf des Tycoon-Systems

Das Tycoon-System wird auf der StP Desktopebene durch Anklicken des in der Abbildung 5.3 dargestellten Piktogramms erreicht.

Abbildung 5.3: DBIS-Tycoon Piktogramm `user_tycoon_32.xbm`

Das so geöffnete Tycoon-Fenster kann auf der Tycoon-Ebene zum Beispiel mit dem Befehl *do exit* wieder terminiert werden.

5.2 Abzubildende Funktionalität der StP Umgebung

Die Tycoon-Bibliothek für den StP Repositoryzugriff muß mindestens die folgenden an sie gestellten Anforderungen befriedigen, welche sich an der Funktionalität des StP Application Program Interface orientieren [IDEStPOMS 94]:

- Generieren und Auffinden von Repositoryobjekten
- Auffinden, Manipulieren und Ausgeben von Attributwerten der spezifizierten Repositoryobjekte
- Verwalten des Hauptspeichers
- Repositorymanagement
- Durchführen von Operationen auf zu einer Einheit zusammengefaßten Objekten innerhalb des StP Repositorys im Rahmen von *Collection* und *Big Collection* Typen
- Interpretieren von OMS Anfragen
- Verwalten und Manipulieren von Datums- und Zeitrepräsentationen
- Initiieren, Überwachen und Abschließen von StP Repositorytransaktionen
- Generieren und Verwalten von ID-Listen
- Durchführen der Fehlerbehandlung bei nicht erfolgreich abgeschlossenen Repositoryoperationen

Diese Anforderungen an den Funktionsumfang der StP Tycoon-Bibliothek werden ihre Berücksichtigung in der Abbildung der Operationen des StP API auf die entsprechenden Tycoon-Funktionen und Module finden.

5.3 Abbildung der StP Repositoryfunktionalität auf die Tycoon-Ebene

Die gesamte Funktionalität für den Repositoryzugriff wird vom StP Application Program Interface zur Verfügung gestellt, welches eine C-basierte Bibliothek von Funktionen und Datentypen darstellt. Auf der Tycoon-Ebene sollen beim Zugriff auf Repositoryobjekte im Rahmen der im Abschnitt 5.4 vorgestellten StP Umgebung, der Tycoon-Bibliothek *stpenv*, ausschließlich Funktionen und Datentypen auf der Basis von TL, der Programmiersprache des Tycoon-Systems, benutzt werden. Somit kann der Tycoon-Anwender unabhängig von StP API C-Funktionsaufrufen auf das StP Repository zugreifen. Diese Vorgehensweise macht es erforderlich, daß innerhalb der Tycoon-Umgebung eine geeignete Abbildung der C-basierten API Funktionen auf adäquate TL-Funktionsdeklarationen erfolgt.

In diesem Abschnitt wird die Abbildung der StP API Funktionalität auf Tycoon-Funktionalität exemplarisch am Beispiel einiger Funktionen des PDM-Typen `node` dargestellt. Diese Vorgehensweise ist auch für alle übrigen PDM-Typen und sonstige innerhalb des Application Program Interface verwendeten Datentypen und Funktionen gültig.

Außer einer einfachen Abbildung der StP API Funktionalität wird die Mächtigkeit der StP Umgebung auf der Tycoon-Seite durch zusätzliche, den Umfang der eigentlichen API Funktionalität erweiternde nützliche Charakteristika ergänzt (siehe Abschnitt 5.4).

5.3.1 Tycoon C-Calls für den StP API-Funktionsaufruf

Das Tycoon-System bietet einen generischen Mechanismus, um auf in externen Programmiersprachen implementierte Funktionalität zugreifen zu können [Mathiske et al. 93]. Eine aufzurufende externe C-Funktion kann mit Hilfe der Tycoon-Funktion *bind* an einen Tycoon-Bezeichner gebunden werden:

let *bspFkt =*
  bind(*:Fun(:Int) :Int "bsp_bibliothek.so" "bsp_funktion" "ii"*)

Hierbei wird die in der dynamischen C-Bibliothek `bsp_bibliothek.so` vorhandene C-Funktion `bsp_funktion` an den Tycoon-Bezeichner *bspFkt* statisch gebunden. Der Wert dieses Bezeichners ist vom Typ ***Fun***(*:Int) :Int*. Innerhalb der Tycoon-Umgebung kann die C-Funktion `bsp_funktion` daraufhin wie folgt indirekt aufgerufen werden:

let *ergebnis :Int = bspFkt(20)*

Welche C-Funktion sich hinter der Tycoon-Funktion *bspFkt* verbirgt, ist für den Tycoon-Benutzer nicht mehr ersichtlich. Diese Vorgehensweise wird im Rahmen des Aufrufs der StP Application Program Interface Funktionen von der Tycoon-Ebene aus im weiteren Verlauf dieser Arbeit verfolgt. Weitere Informationen über die Benutzung externer C-Bibliotheken aus dem Tycoon-System heraus sind in [Mathiske et al. 93] enthalten.

5.3.2 Abbildung von StP API-Funktionen auf Tycoon-Funktionen

Die Abbildung der StP API-Funktionen auf Tycoon-Funktionen erfolgt unter Verwendung der im Abschnitt 5.3.1 dargestellten Vorgehensweise in fünf Schritten:

1. Auf der Systemebene muß als Grundlage für die Tycoon-Funktion *bind* aus den vorhandenen dynamischen StP Bibliotheken eine einzige dynamische Bibliothek erstellt werden, welche die gesamte, benötigte StP Funktionalität des StP API zur Verfügung stellt. Der Name dieser Bibliothek wird beim Anbinden einer StP API Funktion der Tycoon-Funktion *bind* als Parameter übergeben.

   Eine Begründung erfährt das Generieren einer einzigen dynanischen Bibliothek dadurch, daß beim Aufruf einer gebundenen API Funktion eventuell weitere Operationen oder Datentypen aus anderen dynamischen StP Bibliotheken referenziert werden. Um Fehler basierend auf nicht bekanntgegebenen Funktionen oder Datentypen zu vermeiden, werden sämtliche dynamischen Bibliotheken miteinander verbunden. Ein entsprechendes Generierungsskript zum Erstellen der Bibliothek `libstp.so` umfaßt:

   ```
   ld -G -o ${TYCOON_ROOT}/lib/${TYCOON_HOST}/libstp.so
   .../StP/libsparc/libideoms.so .../StP/libsparc/libideparse.so
   .../StP/libsparc/libideeditor.so .../StP/libsparc/libidetools.so
   .../StP/libsparc/libsybdb.so /usr/lib/libm.so
   .../StP/libsparc/libidestub.so /usr/lib/libnsl.so
   ```

2. Eine Voraussetzung für eine erfolgreiche Ausführung der StP API C-Funktionen bzw. einen erfolgreichen Entwicklungsdatenbankzugriff ist ein Eintrag `stp_file_path` mit den entsprechenden Pfaden zu den StP Dateien innerhalb der Datei `ToolInfo`, in welcher grundlegende Initialisierungen einiger StP Systemvariablen und Pfade zu den benötigten Dateien der einzelnen StP Produkte anzugeben sind. Die Datei `ToolInfo` muß für die Benutzung des StP Application Program Interface aus dem Tycoon-System heraus um den folgenden Eintrag ergänzt werden:

   ```
   stp_file_path=.../StP/templates/IDE/omt_booch:
   .../StP/templates/IDE/booch:.../StP/templates/IDE/oo:
   .../StP/templates/IDE/omtcc:.../StP/templates/IDE/t:
   .../StP/templates/IDE/ct:.../StP/templates/IDE
   ```

3. Auf der Tycoon-Ebene müssen die im Abschnitt 4.2.1 wiedergegebenen Systemumgebungsvariablen innerhalb jedes neu gestarteten Tycoon-Subprozesses gesetzt werden. Nur so kann ein erfolgreicher Repositoryzugriff vom Tycoon-System aus erreicht werden.

   Zu diesem Zweck gibt es in der Tycoon-Bibliothek *machineenv* das Modul *uxEnvironment*, mit dessen Funktion *set(name, value :String) :**Ok*** die entsprechenden StP Systemvariablen zu Beginn eines Tycoon-Subprozesses initialisiert werden können. Innerhalb der StP Tycoon-Bibliothek *stpenv* übernimmt die Funktion *omsRepository.init()* diese Aufgabe.

4. Daraufhin können auf der Tycoon-Ebene die *bind*-Funktionen an die entsprechenden Bezeichner gebunden und anschließend wie Tycoon-Funktionen benutzt werden (hier ein Beispiel aus dem Tycoon-Modul *omsNode* der Bibliothek *stpenv*):

```
Let T = omsRep.NodeHandle
let lib = runtimeCore.dynamicLibraryName("stpenv" "libstp")
let handleNil = word.HandleNil(:omsRep.Node)
let error = exception "OMS_ERROR" end

let findCall = bind(:Fun(:String :omsApplicationType.T :omsObjectId.T :String)
                    :T lib "oms_node_find" "siwsw")

let find(rep :omsRepository.T  name :String  type :omsApplicationType.T
         scopeNodeId :omsObjectId.T(T)  signature :String) :T =
  begin
    omsRepository.check(rep)
    let omsNodeTp = findCall(rep name type scopeNodeId signature)
    if word.handleEqual(omsNodeTp handleNil) then
      raise error end
    else
      omsNodeTp
    end
  end
```

   Als Rückgabewert des Aufrufs der C-Funktion `oms_node_find` erhält die Tycoon-Funktion *find* ein 32-Bit Wort, dessen Inhalt dem Zeiger auf die entsprechende C-Struktur `oms_node_tp` entspricht.

5. Der Tycoon-Anwender kann den Zugriff auf das StP Repository nach erfolgter Initialisierung der StP Systemvariablen und einem Import der entsprechenden Module der im Abschnitt 5.4 wiedergegebenen StP Umgebung *stpenv* über die dort deklarierten Tycoon-Funktionen nach dem folgenden Muster durchführen:

```
import omsRepository omsNode

let projdir = "/local/dbis11/software/StP/project"
let system = "tutor"

omsRepository.init()

let rep = omsRepository.connectSystem(projdir system)

let node = omsNode.find(rep "Nodename" omsApplicationType.type
                                  omsObjectId.optional() "Signatur")
...

omsRepository.disconnect(rep)
```

Nach Initialisierung der StP Systemvariablen wird eine Anzahl von Repositoryoperationen stets durch das Öffnen des Repositorys (*omsRepository.connectSystem* oder *omsRepository.connectRep*) und späteres Schließen (*omsRepository.disconnect*) eingerahmt. Zu beachten ist dabei, daß zur Zeit maximal ein Repository geöffnet werden darf.

Ein Beispiel für Tycoon-Anweisungen zum Ausführen der gesamten StP Funktionalität im Kontext mit Repositoryobjekten der StP API Datenstruktur `oms_node_tp` ist im Anhang A.4 wiedergegeben.

5.4 Die Tycoon StP Bibliothek stpenv

Die Bibliothek *stpenv* bietet auf der Tycoon-Ebene die gesamte Funktionalität des StP Application Program Interface, um indirekt unter Verwendung der Funktionalität des StP Object Management Systems auf Repositoryobjekte innerhalb des StP Repositorys zugreifen zu können. Dabei finden die im Abschnitt 5.2 dargestellten Anforderungen an die StP Bibliothek ihre Berücksichtigung in den unterschiedlichen, im Abschnitt 5.4.2 wiedergegebenen, Modulen.

5.4.1 Modellierungsaspekte

Eine Grundlage der Modellierung der Bibliothek *stpenv* stellt das in der Abbildung 3.1 wiedergegebene StP Persistent Data Model dar. Einschränkend gilt aber, daß die „Kategorien“ *CASE Type*, *Annotated Object Type*, *Reference Type* und *SE Type* keine Berücksichtigung im Rahmen der Modellierung der StP Bibliothek finden, da nur ihre Subtypen, die ihrerseits keine Supertypen anderer PDM-Typen sind, im Repository abgelegt werden können.

Desweiteren basiert die Modularisierung der Bibliothek *stpenv* auf den in der Abbildung 3.2 aufgeführten Beziehungen zwischen den einzelnen Datentypen des PDM. Die Art der Ausprägung der Beziehung zwischen zwei Repositoryobjekten ist dabei unterschiedlich zu berücksichtigen. Handelt es sich um eine 1:1 Beziehung, ist dieses durch das Vorhandensein eines betreffenden Attributs des referenzierten Objekts darstellbar. 1:n Beziehungen sind durch die Möglichkeit zur Bildung von „Ansammlungen" referenzierter Objekte, *Collections* oder *Big Collections*, darzustellen. Außerdem sind die sonstigen *CASE Type* Attribute und ihre Verwaltungs- und Manipulationsfunktionen in angemessener Weise zu berücksichtigen. Eine Erläuterung der *Case Type* Attribute und deren Datentypen ist in [IDEStPOMS 94] enthalten.

Die im Abschnitt 3.2 erläuterte Anwendungssicht auf das StP Repository mit ihrer Abbildung von Anwendungsdatentypen auf PDM-Datentypen wurde ebenfalls innerhalb der Implementierung der StP Bibliothek berücksichtigt.

5.4.2 Bibliotheksumfang

In der Abbildung 5.4 ist die Modulstruktur der entwickelten Tycoon StP Umgebung *stpenv* dargestellt. Die Umgebung umfaßt drei verschiedene Arten von Modulen, welche unterschiedliche Funktionalitäten beinhalten:

Spezielle Funktionalität der StP API-Datentypen: Die Module dieser Kategorie stellen im Zusammenhang mit dem Modul *omsGeneric* die Funktionen der StP API C-Datenstrukturen `oms_<pdm_typ>_tp` zur Verfügung. Sie bilden ausschließlich Operationen der C-Datenstrukturen `oms_cntx_tp`, `oms_cntx_ref_tp`, `oms_file_tp`, `oms_file_hist_tp`, `oms_file_lock_tp`, `oms_item_tp`, `oms_link_tp`, `oms_link_ref_tp`, `oms_node_tp`, `oms_node_ref_tp`, `oms_note_tp` und `oms_viewpoint_tp` des StP Application Program Interface ab.

Beispielsweise finden die Operationen des C-basierten StP API PDM-Datentypen `oms_node_tp` ihre ungefähre Entsprechung auf der Tycoon-Ebene im Modul *omsNode*, wobei *omsGeneric* generische Funktionen aller PDM-Datentypen umfaßt. Das Modul *omsGeneric* wird ausschließlich in den Modulen dieser Kategorie importiert. Operationen im Rahmen von *Collection* oder *Big Collection* Typen sind in den Modulen der nächsten Kategorie enthalten.

Allgemeine StP API Funktionalität: Innerhalb dieser Kategorie kann zwischen zwei Arten von Modulen unterschieden werden. Die Module *omsBigCollection* und *omsCollection* stellen generische Funktionen zur Verfügung, mit deren Hilfe *Big Collections* oder *Collections* von mehreren Objekten eines zu spezifizierenden PDM-Datentypen erstellt werden können. Die restlichen Module stellen Funktionen auf den StP API C-Datenstrukturen `oms_app_type_tp`, `oms_idlist_tp`, `oms_pdm_type_tp`, `oms_object_id_tp`, `oms_repos_tp`, `oms_status_tp`, `oms_time_tp` und `oms_txn_tp` bereit.

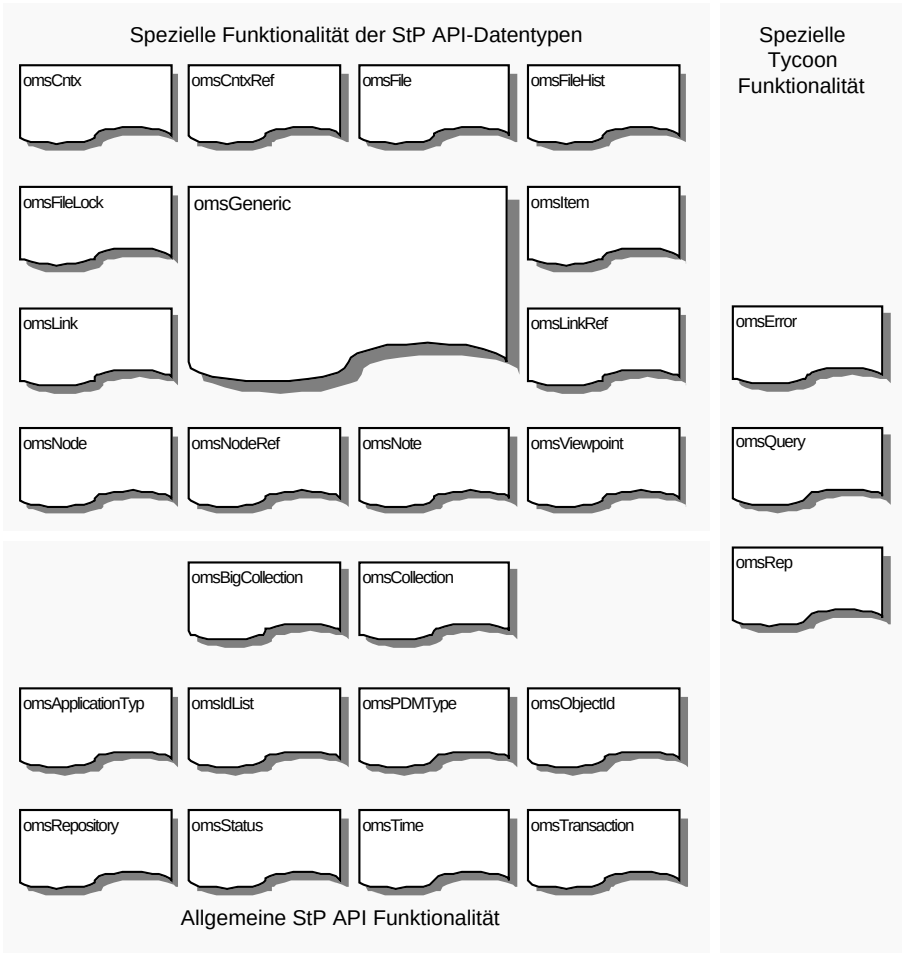

Abbildung 5.4: Tycoon StP Bibliothek *stpenv*

Spezielle Tycoon Funktionalität: Module dieser Kategorie erweitern die StP Funktionalität um eine im Rahmen des StP API nicht vorhandene Funktionalität, welche auf der Tycoon-Ebene benötigt wird, um die Realisierung der Repositoryanbindung überhaupt erstellen zu können.

5.4.3 Aufgaben der Bibliotheksmodule

Grundlage sämtlicher Module der Bibliothek *stpenv* sind die in der Tabelle 5.1 aufgeführten Module *omsError* und *omsRep*. Das Modul *omsQuery* kann bei Bedarf erweitert werden, um beispielsweise korrekte OMS Anfragen in ein Tycoon *String*-Format unzuwandeln.

Modul	Beschreibung
omsError	Spezifikation der auszuführenden Fehlerroutine beim Auftreten eines StP OMS-Fehlers.
omsQuery	Kapselung von StP OMS Anfragen.
omsRep	Enthält die Deklaration der wichtigsten Implementationsdatentypen.

Tabelle 5.1: Module mit Tycoon Funktionalität

Die in der Tabelle 5.2 wiedergegebenen Module entsprechen auf der Ebene des StP Application Program Interface einer Anzahl von Operationen auf C-Datenstrukturen der Bezeichnung `oms_<pdm_typ>_tp`. Das Tycoon-Modul *omsNode* umfaßt zum Beispiel Funktionen des StP API Datentypen `oms_node_tp`.

Modul	Beschreibung
omsGeneric	Generische Hilfsfunktionen, Repositorymanagementfunktionen, Operationen zum Auffinden von Reposiobjekten und Attributzugriffs- und Zuweisungsfunktionen für beliebige/alle PDM-Datentypen.
oms<pdm_typ>	Operationen aus *omsGeneric* und spezielle Funktionen zum Erstellen und Auffinden von Repositoryobjekten und Navigationsoperationen.

Tabelle 5.2: Module mit spezieller PDM-Datentyp Funktionalität

Die weiteren in der Tabelle 5.3 wiedergegebenen Module bilden eine Funktionalität im Rahmen der Operationen auf *CASE Type* Attributen.

Von den Operationen der PDM-Datentypen sind auf der Tycoon-Ebene *Collection* und *Big Collection* Operationen getrennt und in separaten Modulen zusammengefaßt. Auf der C-Ebene sind einige dieser Operationen im Zusammenhang mit den C-Datenstrukturen `oms_coll_tp` und `oms_bigcoll_tp` im Rahmen der Operationen auf Objekten der Datenstruktur `oms_<pdm_typ>_tp` implementiert.

5.5 Realisierungsprobleme

Bezüglich der entwickelten StP Tycoon-Bibliothek *stpenv* sind einige kritische Aspekte anzumerken, welche hauptsächlich einen Einfluß auf die vorgenommene Modellierung der StP Umgebung auf der Ebene des Tycoon-Systems haben. Daraus ergibt sich zwangsläufig, daß

Modul	Beschreibung
omsBigCollection	Generische Operationen zum Generieren und Manipulieren von *Big Collection* Objekten beliebiger/aller PDM-Datentypen.
omsCollection	Generische Operationen zum Generieren und Manipulieren von *Collection* Objekten beliebiger/aller PDM-Datentypen.
omsApplicationType	Operationen entsprechend des StP API Datentyps `oms_app_tp` zur Abbildung von Anwendungsdatentypen auf PDM-Datentypen.
omsIdList	Operationen entsprechend des StP API Datentyps `oms_idlist_tp` zum Verwalten von ID-Listen.
omsPDMType	Operationen entsprechend des StP API Datentyps `oms_pdm_type_tp` zur Verwaltung der internen Darstellung der einzelnen PDM-Datentypen.
omsObjectId	Operationen entsprechend des StP API Datentyps `oms_object_id_tp` zum Erstellen einer individuellen Objekt ID für Objekte innerhalb des StP Repositorys.
omsRepository	Operationen entsprechend des StP API Datentyps `oms_repos_tp` zum Initialisieren der StP Systemvariablen, Öffnen, Verwalten und Schließen des StP Repositorys.
omsStatus	Operationen entsprechend des StP API Datentyps `oms_status_tp` zum Überprüfen ob eine API Operation erfolgreich ausgeführt werden konnte.
omsTime	Operationen entsprechend des StP API Datentyps `oms_time_tp` zur Operation auf Datums- und Zeitrepräsentationen.
omsTransaction	Operationen entsprechend des StP API Datentyps `oms_txn_tp` zum Verwalten von StP Repository Transaktionen.

Tabelle 5.3: Module mit allgemeiner StP API Funktionalität

die gewählte Modellierung auf der Tycoon-Ebene nicht deckungsgleich zu der auf der StP Ebene ist.

Die hauptsächlichen Schwierigkeiten treten bei der Abbildung des StP Persistent Data Models mit Hilfe der StP API C-Funktionen auf Module, Datenstrukturen und Operationen der

Bibliothek *stpenv* und der Interpretation der Anwendungssicht auf das Repository mit der anschließenden Konvertierung der Anwendungsdatentypen auf PDM-Datentypen auf.

5.5.1 Abbildung des StP Persistent Data Models

Das in der Abbildung 3.1 wiedergegebene StP Persistent Data Model stellt den Zusammenhang der existierenden PDM-Datentypen auf der Grundlage von Vererbungsbeziehungen dar, wobei jeder PDM-Datentyp in einer Subtypbeziehung zu einem anderen Typen steht. Desweiteren besitzen alle PDM-Datentypen eine Anzahl von Attributen: spezielle oder, im Rahmen der Vererbungbeziehungen zwischen PDM-Datentypen, gemeinsame Attribute. Leider hat es die Firma IDE jedoch versäumt, diesen Sachverhalt in der Systemdokumentation detailliert wiederzugeben, so daß an dieser Stelle keine weiteren Aussagen über die genaue Ausprägung der Vererbungsbeziehungen getätigt werden können.

Entitäten der PDM-Typen werden in der relationalen Datenbank Sybase abgelegt und sind, wie im Abschnitt 4.2 beschrieben, unter anderem durch den Einsatz der C-Funktionen des StP API zugreifbar. Sowohl auf die gesamte Struktur eines PDM-Objekts als auch auf seine einzelnen Attribute kann zugegriffen werden. Allerdings liefern die API Operationen keine Rückschlüsse über Vererbungscharakteristika referenzierter Objekte, da der Repositoryzugriff mit Hilfe parallel zueinander stehender C-Funktionen realisiert wird.

Da die StP API C-Funktionen keinen Rückschluß zulassen, in welcher Weise das Persistent Data Model auf sie abgebildet worden ist, ist es fraglich, ob man ausgehend von den StP API Operationen im Rahmen eines *Re-Engineering* Vorgehens die tatsächlich vorliegenden Modellierungskomponenten auf der StP Ebene ausdrücken kann, um sie anschließend auf der Tycoon-Ebene zu übernehmen.

5.5.2 Konvertierung von Anwendungs- auf PDM-Datentypen

Wie im Abschnitt 3.2 erläutert, wird die Verbindung zwischen Datentypen von Anwendungsprogrammen und denen des *Persistent Data Models* mit Hilfe der „Anwendungstypendatei“ `app.types` realisiert. Diese Datei enthält jedoch ausschließlich Einträge, die spezifizieren, auf welche CASE Objekttypen die Anwendungsdatentypen abzubilden sind. Die Semantik, die hinter dieser Abbildung steht, ist mit Hilfe der StP API Funktionen nicht nachzuvollziehen und folglich nicht auf die Tycoon-Bibliothek *stpenv* zu übertragen.

So können mit einem grafischen *Editor* zum Beispiel, innerhalb des CASE-Werkzeugs StP/OMT entwickelte Komponenten von Modellen mit Hilfe der Datei `app.types` auf eine Anzahl gültiger Repositoryobjekte abgebildet werden. Die Application Program Interface Operationen leisten diesen Dienst nicht, da sie es nur ermöglichen, isolierte PDM-Objekte im Repository zu generieren oder auf sie zuzugreifen. Eine Modellierung von gültigen Anwendungsdatentypen kann deshalb auf der Tycoon-Ebene durch die API Funktionen nicht erfolgen.

6. Bewertung der Tycoon-Anbindung

Abgesehen davon, daß die entwickelte StP Tycoon Bibliothek *stpenv* die im Abschnitt 5.2 dargestellten Anforderungen hinsichtlich der abzubildenden Funktionalität erfüllt, ergeben sich einige Aspekte, unter deren Berücksichtigung die StP Umgebung *stpenv* zu bewerten ist. Das besondere Augenmerk liegt dabei auf der Interaktion zwischen den Operationen des StP Application Program Interface und dem Tycoon-System.

Da das Tycoon-System die Anforderungen an ein persistentes Objektsystem, wie persistente Datenspeicherung, generische Programmierung und externe Kommunikation, erfüllt [Matthes 93], ist zu bewerten, wie gelungen die einzelnen Aspekte im Zusammenhang mit dem StP API, einem externen Diensterbringer, im Rahmen der Bibliothek *stpenv* berücksichtigt worden sind.

6.1 Persistenz

Die Mächtigkeit des Tycoon-Systems entsteht unter anderem dadurch, daß Programmobjekte als gleichberechtigte Datenobjekte langlebig innerhalb des Tycoon-Objektspeichers abgelegt werden können. Da es sich beim StP Repository prinzipiell ebenfalls um einen persistenten Objektspeicher handelt, liegt der Ansatz nahe, die Persistenzeigenschaften des Repositorys auf die Tycoon-Ebene zu übertragen, indem nach erfolgten Repositoryzugriffen aus dem Tycoon-System heraus die so entstandenen Bindungen innerhalb des Tycoon-Objektspeichers abgelegt werden.

Als problematisch ist diese Vorgehensweise anzusehen, wenn innerhalb des Tycoon-Systems externe C-Funktionen aufgerufen und ausgeführt werden, die als Resultat einen Zeiger auf einen mit dem Ergebnis der C-Funktion assoziierten Hauptspeicherbereich zurückgeben. In der Abbildung 6.1 ist solch ein Funktionsaufruf schematisch dargestellt.

Als Resultat des Tycoon C-Aufrufs der Funktion `oms_node_find(...)` wird ein Zeiger auf eine Hauptspeicheradresse zurückgegeben, in welcher die C-Struktur, hier `oms_node_tp`, abgelegt worden ist. Anschließend kann auf der Tycoon-Ebene mit dem Befehl *do saveSystem* die Bindung des externen Funktionsresultats persistent im Objektspeicher abgelegt werden.

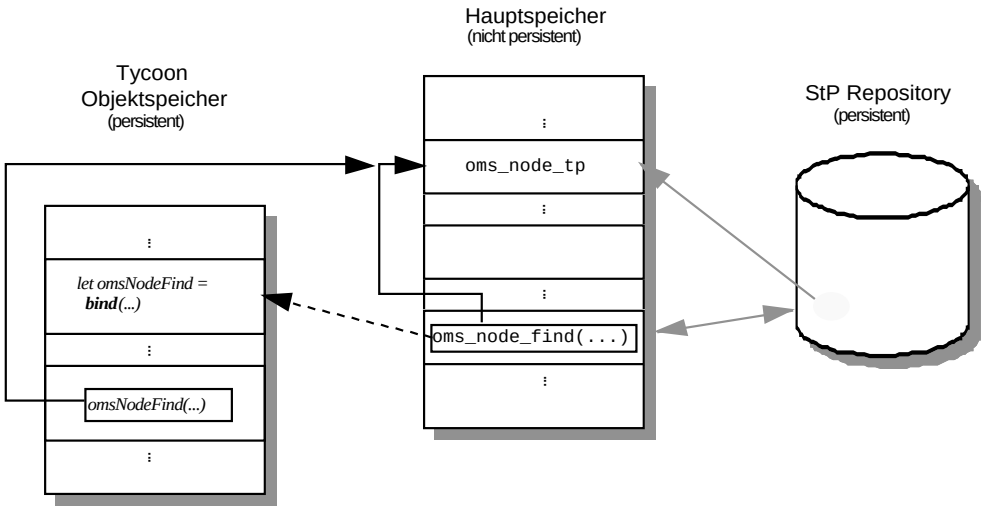

Abbildung 6.1: Verbindung zwischen Tycoon-Objektspeicher und StP Repository

Problematisch ist dabei, daß Adreßbereiche innerhalb des Hauptspeichers nicht als persistent zu betrachten sind. Beim nächstmaligen Referenzieren des mit der Adresse assoziierten Werts aus dem Tycoon-System heraus ist nicht gewährleistet, daß das originale Objekt an der betreffenden Adresse wiederzufinden ist. Die persistente Datenspeicherung innerhalb des Tycoon-Systems steht also im Widerspruch zu der flüchtigen Hauptspeicherablage im Rahmen der Operationen des StP Application Program Interface.

6.2 Typisierung

Die im Tycoon-System vorliegenden strikten Typisierungsmechanismen erlauben, eine gewisse Funktionalität der StP Bibliothek *stpenv* generisch zur Verfügung zu stellen, um Operationen auf PDM-Datentypen, *Collection* und *Big Collection* Typen parametrisiert aufrufen zu können. Beispielsweise ist die Tycoon-Funktion *findById(:omsRepository.T :omsObject.T(E)) :E* im Modul *omsGeneric* für alle PDM-Datentypen gültig.

Desweiteren können Datenstrukturen des StP API auf der Tycoon-Ebene so parametrisiert werden, daß ihr Kontext ersichtlich wird. Ist zum Beispiel auf der C-Ebene ein an eine Funktion übergebener Wert vom Typ `int`, dessen Ursprung in der Struktur `oms_app_type_tp` liegt, so kann dieser Sachverhalt auf der Tycoon-Ebene implementiert werden. Der Funktion *omsNode.create* muß beispielsweise als Parameter ein Wert vom Typ *omsApplicationType.T*, der als Typ *Int* repräsentiert wird, übergeben werden.

Außerdem können Ansammlungen von Objekten von PDM-Datentypen im Rahmen der *Collection* bzw. *Big Collection* Typen objektorientiert dargestellt werden, um ihnen eine

Struktur zu vermitteln. Auf der C-Ebene der Funktionen des StP API stehen die entsprechenden Operationen ausschließlich parallel nebeneinander.

6.3 Flexibilität

Das Tycoon-System als ein offenes System ermöglicht in gewissen Grenzen den Zugriff auf Leistungen externer Diensterbringer. Desweiteren ist es externen Anwendungen möglich, Funktionen des Tycoon-Systems zu aktivieren, um Aktionen basierend auf dem persistenten Zustand des Objektsystems zu steuern [Matthes 93].

Im Rahmen der Umgebung *stpenv* kann von externen Diensten indirekt auf das StP Repositoy zugegriffen werden. Umgekehrt ist es möglich, aus dem Tycoon-System heraus einen externen Dienst zu nutzen, um nach erfolgtem Repositoryzugriff beispielsweise die Ergebnisse fensterorientiert, basierend auf der Funktionalität von StarView, darzustellen. Die StP Bibliothek *stpenv* fungiert also als *Gateway* zwischen dem StP Object Management System und weiteren externen Diensten, wobei die Verbindung bidirektional zu interpretieren ist.

6.4 Wartbarkeit

Der Begriff der offenen Systemarchitektur bringt automatisch das Problem der Wartbarkeit mit sich. In diesem Zusammenhang muß festgestellt werden, daß die Wartbarkeit der StP Umgebung *stpenv* von zwei Seiten determiniert wird.

Ein hoher Wartungsaufwand entsteht, wenn auf der C-Ebene im Rahmen des StP API Funktionsdeklarationen verändert werden. Im einfachsten Fall müßten nur die betreffenden Tycoon C-Aufrufe verifiziert werden. Allerdings entsteht ein hoher Aufwand, wenn eine Änderung der StP API Funktionssignaturen eine manuelle Umgestaltung der Typisierung der Tycoon-Funktionen erforderlich macht.

Nicht so schwerwiegend ist eine Änderung der TL-Syntax. Einzig ist dabei zu beachten, daß ein Aufruf der externen C-Funktionen des StP Application Program Interface nach dem vorhandenen Schema möglich ist.

7. Zusammenfassung und Ausblick

Als Hauptaufgabe dieser Arbeit sollte innerhalb des Tycoon-Systems eine Bibliothek erstellt werden, mit deren Hilfe auf die gesamten im StP Repository abgelegten Modellierungsdaten der StP CASE-Werkzeuge aus dem Tycoon-System heraus zugegriffen werden kann. Als Resultat dieser Arbeit kann das Tycoon-System als integrierte Entwicklungsumgebung innerhalb der StP Umgebung angesehen werden, der Repositoryzugriff kann mit Hilfe der Funktionen der Tycoon-Bibliothek *stpenv* durchgeführt werden.

Auf dem Weg zur erfolgreichen Umsetzung der an diese Arbeit gestellten Anforderungen wurden im Kapitel 4 die unterschiedlichen Möglichkeiten des Entwicklungsdatenbankzugriffs untersucht und bewertet, inwieweit sie zur späteren Benutzung innerhalb des Tycoon-Systems befähigt sind. Es stellte sich heraus, daß zur Anbindung des Tycoon-Systems an die CASE-Umgebung StP über die Entwicklungsdatenbank die Benutzung des StP Application Program Interface aus dem Tycoon-System heraus der Benutzung von StP QRL Skripten vorzuziehen ist. Folglich stellt das StP API die Schnittstelle zwischen dem StP Object Management System und den Funktionen der Tycoon-Bibiliothek *stpenv* dar.

Basierend auf diesen Erkenntnissen ist die Tycoon-Bibliothek *stpenv* entwickelt und implementiert worden. Sie bietet unter Beachtung der im Abschnitt 5.3.2 dargestellten Vorgehensweise die im Abschnitt 5.2 dargestellte Funktionalität für den Repositoryzugriff. Desweiteren ist das gesamte Tycoon-System als Entwicklungsumgebung in die StP Umgebung integriert und im StP Desktop aufrufbar. Zu berücksichtigen sind allerdings die im Kapitel 6 aufgeführten und zu beachtenden Aspekte im Zusammenhang mit der Benutzung der Funktionen der Bibliothek *stpenv*.

Zukünftige Lösungsansätze der im Abschnitt 6.1 dargestellten Persistenzproblematik könnten auf der Basis der Operationen des Tycoon-Moduls *volatile* gelöst werden, welches die Verwaltung von flüchtigen Daten in der persistenten Umgebung des Tycoon-Systems ermöglicht.

Eine Verbesserung der Wartbarkeit könnte durch einen Generator zur Anbindung externer C-Funktionen erreicht werden, welcher eine parametrisierbare Schnittstelle zwischen den Funktionen des StP Application Program Interface und den Tycoon C-Aufrufen bildet. Ein ähnlicher Ansatz ist für die Anbindung externer C++-Bibliotheken in [Geisler 95] dargestellt, welcher aber im Rahmen dieser Arbeit nicht weiter verfolgt werden kann, da die den

StP API C-Funktionen zugrundeliegende originale C++-Modellierung nicht vorliegt.

Eine Erweiterung der Tycoon-Bibliothek *stpenv* könnte erreicht werden, indem man im Rahmen des Moduls *omsApplicationType* versucht, eine gültige Abbildung von Anwendungsdatentypen auf PDM-Datentypen ähnlich der Vorgehensweise im Rahmen der Abbildungsdatei `app.types` zu generieren. Auf diese Weise könnte man nicht nur Objekte der Subtypen des *Persistent Data Models* im StP Repository ablegen sondern auch *CASE Type* Objekte. Desweiteren wäre es möglich, im Modul *omsQuery* OMS Anfragen im Tycoon-Format vom Typ *String* zu erzeugen bzw. zu überprüfen, inwieweit eine Anfrage vom Tycoon-Datentyp *String* einer syntaktisch korrekten OMS Anfrage entspricht.

Desweiteren könnte man versuchen, auf der Basis mehrerer Tycoon-Subprozesse eine Kommunikation zwischen mehreren Repositorys zu erreichen, obwohl innerhalb eines Tycoon-Subprozesses nur ein Repository zur Zeit geöffnet sein darf. Diese Einschränkung kann so eventuell umgangen werden.

Abschließend kann festgehalten werden, daß die implementierte StP Tycoon Bibliothek *stpenv* die gesamte StP Application Program Interface Funktionalität unter Ausnutzung der Vorteile des Tycoon-Systems beinhaltet. Die Entwicklung einer Animations- und Simulationskomponente auf der Tycoon-Ebene zur Animation und Simulation der mit dem CASE-Werkzeug StP/OMT grafisch entwickelten Prozeßbeschreibungen zwecks Entwurfsvalidierung kann auf dieser Arbeit aufbauend durchgeführt werden.

A. Dateiauflistung

A.1 StP QRL-Skript Select.qrl

A.1.1 Skriptdatei Select.qrl

```
/* ------------------------------------------------------------------- */
/*      Datei:  Select.qrl                                             */
/*      Autor:  Oliver Nietsch                                         */
/*                                                                     */
/*      Universitaet Hamburg                                           */
/*      Fachbereich Informatik                                         */
/*      Arbeitsbereich DBIS                                            */
/*                                                                     */
/*      Inhalt: StP OMS-Anfragen fuer den Zugriff auf Objekte im StP   */
/*              Repository und zur Ausgabe ihrer Attributwerte         */
/*              hier:   nur exemplarischer Skriptinhalt auf Basis      */
/*                      der StP OMS Schnittstellenfunktionen           */
/*              ProjDir: /local/dbis11/software/StP/Examples           */
/*              System:  email                                         */
/*                                                                     */
/*      Version vom  31. 8.1995                                        */
/* ------------------------------------------------------------------- */

void
main()

{

        // Variablendeklaration fuer alle vorgesehenen PDM-Typen:

        cntx_ref        cntx_ref_var;
        cntx            cntx_var;
        file_hist       file_hist_var;
        file_lock       file_lock_var;
        file            file_var;
        item            item_var;
        link_ref        link_ref_var;
        link            link_var;
        node_ref        node_ref_var;
        node            node_var;
        note            note_var;
        viewpoint       viewpoint_var;
```

```
// Anfragen fuer den Zugriff und zur Ausgabe einiger im
// StP Repository abgelegter Werte der PDM-Attribute


//      1. for_each_in_select Funktion:
//      -------------------------------

for_each_in_select("cntx_ref", cntx_ref_var)
print_line("cntx_ref_id: \"" + cntx_ref_var.id + "\" cntx_id: " +
        cntx_ref_var.cntx_id);

for_each_in_select("cntx", cntx_var)
print_line("cntx_id: \"" + cntx_var.id + "\" cntx_name: " +
        cntx_var.name + "\" cntx_type: " + cntx_var.type);

for_each_in_select("file_hist[id >= 59302]", file_hist_var)
print_line("file_hist_id: \"" + file_hist_var.id + "\" file_id: " +
        file_hist_var.file_id + "\" file_hist_type: "
        + file_hist_var.type + "\" file_hist_user: "
        + file_hist_var.user);

for_each_in_select("file_lock", file_lock_var)
print_line("file_lock_id: \"" + file_lock_var.id
        + "\" file_lock_file_id: " + file_lock_var.file_id
        + "\" file_lock_time: " + file_lock_var.time);

for_each_in_select("file[id = 223]", file_var)
print_line("file_id: \"" + file_var.id
        + "\" file_name: " + file_var.name
        + "\" file-annot_file_id: " + file_var.annot_file_id);

for_each_in_select("item[id >= 55183]", item_var)
print_line("item_id: \"" + item_var.id
        + "\" item_note_id: " + item_var.note_id
        + "\" item_obj_id: " + item_var.obj_id);

for_each_in_select("link_ref[id >= 54300]", link_ref_var)
print_line("link_ref_id: \"" + link_ref_var.id
        + "\" link_ref_link_id : " + link_ref_var.link_id
        + "\" from_node_ref_id: " + link_ref_var.from_node_ref_id
        + "\" to_node_ref_id: "+ link_ref_var.to_node_ref_id);

for_each_in_select("link[id >= 54400]", link_var)
print_line("link_id: \"" + link_var.id
        + "\" from_node_id: " + link_var.from_node_id
        + "\" to_node_id: " + link_var.to_node_id);

for_each_in_select("node_ref[id >= 57000]", node_ref_var)
print_line("node_ref_id: \"" + node_ref_var.id
        + "\" node_id: \"" + node_ref_var.node_id
        + "\" file_id: " + node_ref_var.file_id);

for_each_in_select("node[type == OMTEvent]", node_var)
print_line("Der Node \"" + node_var.name + "\" ist" +
        " vom Typ " + node_var.type + ", node_id: " +
        node_var.id + ", annot_file_id: " + node_var.annot_file_id);

for_each_in_select("note[type = OMTUseCaseDefinition]", note_var)
print_line("note_id: \"" + note_var.id
        + "\" note_type : " + note_var.type
```

```
        + "\" note_name: " + note_var.name);

for_each_in_select("viewpoint[id <= 12299]", viewpoint_var)
print_line("viewpoint_id: \"" + viewpoint_var.id
        + "\" node_id : " + viewpoint_var.node_id
        + "\" viewpoint_type: " + viewpoint_var.type
        + "\" node_ref_id: "+ viewpoint_var.node_ref_id);


//      2. find_by_query Funktion:
//      --------------------------

int id = 14209;

print(find_by_query("node[id == "+id+"]"));


//      3. selection_count Funktion:
//      ----------------------------

string query;
int zaehler;

query = "node[type == OMTEvent && annot_file_id == 0]";
zaehler = selection_count(query);

print("Anzahl gefundener Objekte, die vom Type OMTEvent sind" +
        " und den Wert 0 als annot_file_id besitzen: " + zaehler);


//      4. app_type_print_string Funktion: nicht beruecksichtigt.
//      ---------------------------------


//      5. id_list_create Funktion:
//      ---------------------------

node aktueller_node;

query = "node[OMTEvent]";

id_list_create(query, "node_liste");
for_each_in_select("node[node_liste]", aktueller_node)
print_line(aktueller_node.type + " " + aktueller_node.name);

query = "node[node_liste && name$'O*']";

for_each_in_select(query, aktueller_node)
print_line(aktueller_node.type + " " + aktueller_node.name);


//      6. id_list_free Funktion:
//      -------------------------

id_list_free("node_liste");


//      7. list_select Funktion:
//      ------------------------
```

```
        list meine_liste;

        meine_liste = list_select("node[OMTEvent && 14209 || 14211]");
        print(meine_liste);


        //      8. set_select Funktion: siehe 7., list_select_Funktion
        //      -----------------------


        //      9. to_oms_string Funktion: nicht beruecksichtigt.
        //      --------------------------

}
```

A.1.2 Aufbereitete Skriptausgabe Select_Output.txt

```
/* ------------------------------------------------------------------- */
/*      Datei:  Select_Output.txt                                      */
/*      Autor:  Oliver Nietsch                                         */
/*                                                                     */
/*      Universitaet Hamburg                                           */
/*      Fachbereich Informatik                                         */
/*      Arbeitsbereich DBIS                                            */
/*                                                                     */
/*      Inhalt: Ausgabe des QRL Skripts "Select.qrl"                   */
/*              Ausfuehrung: qrp Select.qrl                            */
/*              ProjDir: /local/dbis11/software/StP/Examples           */
/*              System:  email                                         */
/*                                                                     */
/*      Version vom  31. 8.1995                                        */
/* ------------------------------------------------------------------- */

1. for_each_in_select Funktion:
================================

cntx_ref_id: "54602" cntx_id: 54601
cntx_ref_id: "54604" cntx_id: 54603

cntx_id: "54601" cntx_name: start input" cntx_type: OMTEventStart
cntx_id: "54603" cntx_name: <= start input + 5 minutes" cntx_type: OMTEventConstraintBefore

file_hist_id: "59302" file_id: 231" file_hist_type: FileView" file_hist_user: gschroed
file_hist_id: "59401" file_id: 223" file_hist_type: FileView" file_hist_user: gschroed
file_hist_id: "59402" file_id: 223" file_hist_type: FileView" file_hist_user: gschroed
file_hist_id: "59501" file_id: 73" file_hist_type: FileView" file_hist_user: gschroed

file_lock: nicht vorhanden.

file_id: "223" file_name: Mailbox" file-annot_file_id: 0

item_id: "55183" item_note_id: 55182" item_obj_id: 4601
item_id: "55188" item_note_id: 32202" item_obj_id: 3701
item_id: "55190" item_note_id: 55189" item_obj_id: 3701

link_ref_id: "54301" link_ref_link_id : 54206" from_node_ref_id: 37302" to_node_ref_id: 37301
link_ref_id: "54402" link_ref_link_id : 53703" from_node_ref_id: 15004" to_node_ref_id: 15005
link_ref_id: "54404" link_ref_link_id : 54403" from_node_ref_id: 15004" to_node_ref_id: 15005

link_id: "54403" from_node_id: 402" to_node_id: 301
link_id: "59202" from_node_id: 4001" to_node_id: 401
link_id: "59203" from_node_id: 401" to_node_id: 4001

node_ref_id: "57001" node_id: "7202" file_id: 141
node_ref_id: "57002" node_id: "7204" file_id: 141
node_ref_id: "57003" node_id: "32304" file_id: 141
node_ref_id: "57004" node_id: "3901" file_id: 141

Der Node "send message request" ist vom Typ OMTEvent, node_id: 14209, annot_file_id: 0
Der Node "bad receiver" ist vom Typ OMTEvent, node_id: 14211, annot_file_id: 0
Der Node "create message" ist vom Typ OMTEvent, node_id: 15010, annot_file_id: 0
Der Node "good receiver" ist vom Typ OMTEvent, node_id: 15017, annot_file_id: 0
Der Node "delete message" ist vom Typ OMTEvent, node_id: 15302, annot_file_id: 0
Der Node "create user" ist vom Typ OMTEvent, node_id: 16107, annot_file_id: 0
Der Node "duplicate user" ist vom Typ OMTEvent, node_id: 16109, annot_file_id: 0
```

```
Der Node "create message request" ist vom Typ OMTEvent, node_id: 16402, annot_file_id: 0
Der Node "create new user" ist vom Typ OMTEvent, node_id: 16508, annot_file_id: 0
Der Node "create conference" ist vom Typ OMTEvent, node_id: 16707, annot_file_id: 0
Der Node "duplicate conference" ist vom Typ OMTEvent, node_id: 16709, annot_file_id: 0
Der Node "delete user or conference" ist vom Typ OMTEvent, node_id: 17007, annot_file_id: 0
Der Node "bad user or conference" ist vom Typ OMTEvent, node_id: 17009, annot_file_id: 0
Der Node "delete user" ist vom Typ OMTEvent, node_id: 17108, annot_file_id: 0
Der Node "no messages" ist vom Typ OMTEvent, node_id: 17409, annot_file_id: 0
Der Node "bad message number" ist vom Typ OMTEvent, node_id: 17512, annot_file_id: 0
Der Node "show header" ist vom Typ OMTEvent, node_id: 17611, annot_file_id: 0
Der Node "check for empty" ist vom Typ OMTEvent, node_id: 17809, annot_file_id: 0
Der Node "mailbox empty" ist vom Typ OMTEvent, node_id: 17812, annot_file_id: 0
Der Node "messages available" ist vom Typ OMTEvent, node_id: 18112, annot_file_id: 0
Der Node "bad conference" ist vom Typ OMTEvent, node_id: 18209, annot_file_id: 0
Der Node "join conference request" ist vom Typ OMTEvent, node_id: 18306, annot_file_id: 0
Der Node "good conference" ist vom Typ OMTEvent, node_id: 18310, annot_file_id: 0
Der Node "bad user" ist vom Typ OMTEvent, node_id: 18506, annot_file_id: 0
Der Node "login user request" ist vom Typ OMTEvent, node_id: 18601, annot_file_id: 0
Der Node "start" ist vom Typ OMTEvent, node_id: 18706, annot_file_id: 0
Der Node "quit request" ist vom Typ OMTEvent, node_id: 18809, annot_file_id: 0
Der Node "user logged out" ist vom Typ OMTEvent, node_id: 18903, annot_file_id: 0
Der Node "logout request" ist vom Typ OMTEvent, node_id: 18905, annot_file_id: 0
Der Node "print" ist vom Typ OMTEvent, node_id: 25101, annot_file_id: 0
Der Node "incoming message" ist vom Typ OMTEvent, node_id: 41601, annot_file_id: 0
Der Node "create new conference" ist vom Typ OMTEvent, node_id: 51801, annot_file_id: 0
Der Node "delete conference" ist vom Typ OMTEvent, node_id: 52001, annot_file_id: 0
Der Node "print selected message" ist vom Typ OMTEvent, node_id: 52101, annot_file_id: 0
Der Node "list message headers" ist vom Typ OMTEvent, node_id: 52103, annot_file_id: 0
Der Node "send message" ist vom Typ OMTEvent, node_id: 52401, annot_file_id: 0
Der Node "enter body" ist vom Typ OMTEvent, node_id: 52403, annot_file_id: 0
Der Node "read message" ist vom Typ OMTEvent, node_id: 52601, annot_file_id: 0
Der Node "check conference" ist vom Typ OMTEvent, node_id: 53401, annot_file_id: 0
Der Node "add to mailbox" ist vom Typ OMTEvent, node_id: 53601, annot_file_id: 0
Der Node "check user or conference" ist vom Typ OMTEvent, node_id: 53701, annot_file_id: 0

note_id: "38175" note_type : OMTUseCaseDefinition" note_name:
note_id: "48725" note_type : OMTUseCaseDefinition" note_name:

viewpoint_id: "12047" node_id : 301" viewpoint_type: AllAggregation" node_ref_id: 12001
viewpoint_id: "12291" node_id : 301" viewpoint_type: AllOperations" node_ref_id: 12117
viewpoint_id: "12292" node_id : 301" viewpoint_type: SubsystemMember" node_ref_id: 12117
viewpoint_id: "12048" node_id : 401" viewpoint_type: AllGeneralization" node_ref_id: 12002
viewpoint_id: "12297" node_id : 401" viewpoint_type: AllOperations" node_ref_id: 12120
viewpoint_id: "12298" node_id : 401" viewpoint_type: SubsystemMember" node_ref_id: 12120
viewpoint_id: "12299" node_id : 402" viewpoint_type: AllOperations" node_ref_id: 12121
viewpoint_id: "12289" node_id : 2501" viewpoint_type: AllOperations" node_ref_id: 12116
viewpoint_id: "12290" node_id : 2501" viewpoint_type: SubsystemMember" node_ref_id: 12116
viewpoint_id: "12295" node_id : 2601" viewpoint_type: AllOperations" node_ref_id: 12119
viewpoint_id: "12296" node_id : 2601" viewpoint_type: SubsystemMember" node_ref_id: 12119
viewpoint_id: "12293" node_id : 3101" viewpoint_type: AllOperations" node_ref_id: 12118
viewpoint_id: "12294" node_id : 3101" viewpoint_type: SubsystemMember" node_ref_id: 12118
viewpoint_id: "12287" node_id : 3701" viewpoint_type: AllOperations" node_ref_id: 12109
viewpoint_id: "12288" node_id : 3701" viewpoint_type: SubsystemMember" node_ref_id: 12109
viewpoint_id: "12049" node_id : 4001" viewpoint_type: AllAggregation" node_ref_id: 12008
viewpoint_id: "12285" node_id : 4001" viewpoint_type: AllOperations" node_ref_id: 12108
viewpoint_id: "12286" node_id : 4001" viewpoint_type: SubsystemMember" node_ref_id: 12108
viewpoint_id: "12283" node_id : 4601" viewpoint_type: AllOperations" node_ref_id: 12107
viewpoint_id: "12284" node_id : 4601" viewpoint_type: SubsystemMember" node_ref_id: 12107
viewpoint_id: "12281" node_id : 6301" viewpoint_type: AllOperations" node_ref_id: 12106
viewpoint_id: "12282" node_id : 6301" viewpoint_type: SubsystemMember" node_ref_id: 12106
```

```
viewpoint_id: "12279" node_id : 8101" viewpoint_type: AllOperations" node_ref_id: 12105
viewpoint_id: "12280" node_id : 8101" viewpoint_type: SubsystemMember" node_ref_id: 12105

2. find_by_query_Funktion:
==========================

{
 id = 14209
 name = send message request
 type = OMTEvent
 scope_node_id = 0
 sig =
 annot_file_id = 0
}

3. selection_count Funktion:
============================

Anzahl gefundener Objekte, die vom Type OMTEvent sind und den Wert 0 als annot_file_id besitzen: 41

5. id_list_create Funktion:
===========================

OMTEvent send message request
OMTEvent bad receiver
OMTEvent create message
OMTEvent good receiver
OMTEvent delete message
OMTEvent create user
OMTEvent duplicate user
OMTEvent create message request
OMTEvent create new user
OMTEvent create conference
OMTEvent duplicate conference
OMTEvent delete user or conference
OMTEvent bad user or conference
OMTEvent delete user
OMTEvent no messages
OMTEvent bad message number
OMTEvent show header
OMTEvent check for empty
OMTEvent mailbox empty
OMTEvent messages available
OMTEvent bad conference
OMTEvent join conference request
OMTEvent good conference
OMTEvent bad user
OMTEvent login user request
OMTEvent start
OMTEvent quit request
OMTEvent user logged out
OMTEvent logout request
OMTEvent print
OMTEvent incoming message
OMTEvent create new conference
OMTEvent delete conference
OMTEvent print selected message
OMTEvent list message headers
OMTEvent send message
OMTEvent enter body
```

```
OMTEvent read message
OMTEvent check conference
OMTEvent add to mailbox
OMTEvent check user or conference

7. list_select Funktion:
========================

{
 0 =
    {
     id = 14209
     name = send message request
     type = OMTEvent
     scope_node_id = 0
     sig =
     annot_file_id = 0
    }
 1 =
    {
     id = 14211
     name = bad receiver
     type = OMTEvent
     scope_node_id = 0
     sig =
     annot_file_id = 0
    }
}
```

A.2 C-Programm api3.c

A.2.1 Programmdatei api3.c

```
/* ------------------------------------------------------------------- */
/*      Datei:  api3.c                                                 */
/*      Autor:  Oliver Nietsch                                         */
/*                                                                     */
/*      Universitaet Hamburg                                           */
/*      Fachbereich Informatik                                         */
/*      Arbeitsbereich DBIS                                            */
/*                                                                     */
/*      INHALT: Aufruf initialisierter StP API Funktionen aus diesem   */
/*              C-Programm heraus um auf StP Repositoryobjekte         */
/*              zuzugreifen und deren Attributwerte auszugeben.        */
/*              hier: nur exemplarischer Funktionsaufruf einiger,      */
/*                    wichtiger ausgewaehlter StP API Funktionen       */
/*              ProjDir: /local/dbis11/software/StP/project            */
/*              System:  tutor (wg. rw-Rechte)                         */
/*                                                                     */
/*      Version vom   5.10.1995                                        */
/* ------------------------------------------------------------------- */

#include <stdio.h>
#include "oms_stdinc.h"


void
printStatus(oms_status_tp status)
{
  if (status == OMS_FAIL) printf("Zugriff auf Repository nicht moeglich\n");
  else if (status == OMS_SUCCEED) printf("Zugriff auf Repository erfolgt.\n");
  else if (status == OMS_ABORT) printf("Zugriff auf Repository abgebrochen.\n");
  else printf("Unbekannter Fehler.\n");
}


void
printName(char *name)
{
  if (! name)
    printf("Kein Name angegeben\n");
  else
    printf("%s\n", name);
}


void
main()
{
  char                  *name1, *name2, *name3;
  char                  *name = "interessanter Node";
  oms_app_type_tp       type = 701576;      /* int, s. oms.app.types
                                               hier: "OMTEvent" */
  oms_object_id_tp      scope = 0;          /* long */
  char                  *sig = "moin_moin";
  oms_object_id_tp      annot = 7;          /* long */
  FILE                  *fp;
```

```
oms_node_tp             *node1, *node2, *node3, *node4;
char                    buf[1024];
oms_status_tp           status;


printf("\n");

status = oms_sys_repos_open("/local/dbis11/software/StP/project", "tutor");
printStatus(status);

name1 = oms_repos_current_server_name();
printf("Server Name: ");
printName(name1);

name2 = oms_repos_current_rep_name();
printf("System Name: ");
printName(name2);


node1 = oms_node_find(name, type, scope, sig);
if (! node1)
  {
    printf("Node nicht vorhanden bzw. nicht aufgefunden.\n");
    printf("Generiere neuen Node mit den angegebenen Attributwerten.\n");
    printf("Setze 1.Pointer auf erstellten Node.\n");
    node1 = oms_node_create(name, type, scope, sig);
    if (! node1)
      printf("Kann neuen Node nicht erstellen.\n");
  }
else
  {
    printf("Angegebener Node ist schon vorhanden -> Keine Neugenerierung.\n");
    printf("1.Pointer auf vorhandenen Node gesetzt.\n");
  }

node2 = oms_node_find(name, type, scope, sig);
if (! node2)
  printf("Kann vorhandenen Node nicht finden.\n");
else
  printf("Vorhandenen Node wiedergefunden und 2.Pointer darauf gesetzt.\n");


node3 = oms_node_find_by_id(node2 -> id);
if (! node3)
  printf("Kann vorhandenen Node nicht finden.\n");
else
  printf("Node nochmals aufgefunden und 3.Pointer darauf gesetzt.\n");


sprintf(buf, "node[name='%s' && type=%d && scope_node_id=%ld && sig='%s']", name, type, scope, sig);
node4 = oms_node_find_by_query(buf);
if (node4)
  {
    printf("In Query angegebenen Node im Repository wiedergefunden.\n");
    printf("-> 4.Pointer auf gefundenen Node gesetzt.\n");
  }
else
  {
    printf("Query negativ. Node nicht vorhanden/ aufgefunden.\n");
    printf("4.Pointer nicht gesetzt.\n");
```

```
    }

  printf("Node-Attribute (1.Pointer):\n");
  oms_node_print(node1, fp);
  printf("Node-Attribute (4.Pointer):\n");
  oms_node_print(node4, fp);


  oms_node_free(&node4);
  printf("4.Pointer freigegeben (nur Hauptspeicher-Operation).\n");


  status = oms_node_update(node1);
    if (status != OMS_SUCCEED)
      printf("Repository-Update (1.Pointer) nicht erfolgt.\n");
    else
      {
        printf("Repository-Update erfolgt.\n");
        printf("Node (1.Pointer) im Repository persistent abgelegt.\n");
      }


  oms_node_annot_file_id_asgn(node1, 0);
  status = oms_node_update(node1);
    if (status != OMS_SUCCEED)
      printf("Repository-Update (1.Pointer) nicht erfolgt.\n");
    else
      {
        printf("Wert annot_file_id auf 0 gesetzt.\n");
        printf("Repository-Update erfolgt.\n");
        printf("Node (1.Pointer) im Repository persistent abgelegt.\n");
      }


  printf("Node-Attribute (1.Pointer) nach Wert-Aenderung der annot_file_id:\n");
  node1 = oms_node_find(name, type, scope, sig);
  oms_node_print(node1, fp);


  status = oms_repos_close();
  printStatus(status);

}
```

A.2.2 Aufbereitete Programmausgabe api3_Output.txt

```
/* ------------------------------------------------------------------- */
/*      Datei:  api3_Output.txt                                        */
/*      Autor:  Oliver Nietsch                                         */
/*                                                                     */
/*      Universitaet Hamburg                                           */
/*      Fachbereich Informatik                                         */
/*      Arbeitsbereich DBIS                                            */
/*                                                                     */
/*      INHALT: Aufbereitete Ausgabe des C-Programms "api3.c"          */
/*              Ausfuehrung: api3                                      */
/*              ProjDir: /local/dbis11/software/StP/project            */
/*              System:  tutor (wg. rw-Rechte)                         */
/*                                                                     */
/*      Version vom   5.10.1995                                        */
/* ------------------------------------------------------------------- */

Wertinitialisierung (Node Objekt ist im Repository schon vorhanden):
====================================================================

*name =         "ausgedachter Node";    Node Name
type =          701576;                 hier: Node Typ:OMTEvent
scope =         0;                      scope_node_id
*sig =          "hallo";                signature
annot =         5;                      annot_file_id


Ausgabe:
========

Zugriff auf Repository erfolgt.
Server Name: STP_SERVER
System Name: tutor

Angegebener Node ist schon vorhanden -> Keine Neugenerierung.
1.Pointer auf vorhandenen Node gesetzt.

Vorhandenen Node wiedergefunden und 2.Pointer darauf gesetzt.

Node nochmals aufgefunden und 3.Pointer darauf gesetzt.

In Query angegebenen Node im Repository wiedergefunden.
-> 4.Pointer auf gefundenen Node gesetzt.

Node-Attribute (1.Pointer):
id = 13803
name = ausgedachter Node
type = OMTEvent
scope_node_id = 0
sig = hallo
annot_file_id = 5

Node-Attribute (4.Pointer):
id = 13803
name = ausgedachter Node
type = OMTEvent
scope_node_id = 0
sig = hallo
annot_file_id = 5
```

```
4.Pointer freigegeben (nur Hauptspeicher-Operation).

Repository-Update erfolgt.
Node (1.Pointer) im Repository persistent abgelegt.

Wert annot_file_id auf 0 gesetzt.
Repository-Update erfolgt.
Node (1.Pointer) im Repository persistent abgelegt.

Node-Attribute (1.Pointer) nach Wert-Aenderung der annot_file_id:
id = 13803
name = ausgedachter Node
type = OMTEvent
scope_node_id = 0
sig = hallo
annot_file_id = 0

Zugriff auf Repository erfolgt.


Wertinitialisierung (Node Objekt ist im Repository nicht vorhanden):
====================================================================

*name =         "interessanter Node";   Node Name
type =          701576;                 hier: Node Typ:OMTEvent
scope =         0;                      scope_node_id
*sig =          "moin_moin";            signature
annot =         7;                      annot_file_id


Ausgabe:
========

Zugriff auf Repository erfolgt.
Server Name: STP_SERVER
System Name: tutor

Node nicht vorhanden bzw. nicht aufgefunden.
Generiere neuen Node mit den angegebenen Attributwerten.
Setze 1.Pointer auf erstellten Node.

Vorhandenen Node wiedergefunden und 2.Pointer darauf gesetzt.

Node nochmals aufgefunden und 3.Pointer darauf gesetzt.

Query negativ. Node nicht vorhanden/ aufgefunden.
4.Pointer nicht gesetzt.

Node-Attribute (1.Pointer):
id = 14103
name = interessanter Node
type = OMTEvent
scope_node_id = 0
sig = moin_moin
annot_file_id = 7

Node-Attribute (4.Pointer):
4.Pointer freigegeben (nur Hauptspeicher-Operation).
```

```
Repository-Update erfolgt.
Node (1.Pointer) im Repository persistent abgelegt.

Wert annot_file_id auf 0 gesetzt.
Repository-Update erfolgt.
Node (1.Pointer) im Repository persistent abgelegt.

Node-Attribute (1.Pointer) nach Wert-Aenderung der annot_file_id:
id = 14103
name = interessanter Node
type = OMTEvent
scope_node_id = 0
sig = moin_moin
annot_file_id = 0

Zugriff auf Repository erfolgt.
```

A.3 Tycoon-Startskript start*

```
#!/bin/sh
# File: .../bin/teamware
# Author: Andreas Gawecki
# Date: 08-FEB-1994
# Purpose: 'activate' a teamware workspace
# Updates: 17-Feb-1995 AG,AR: CODEMGR_DIR_FLP
#          13-Okt-1995 Oliver Nietsch fuer Privatgebrauch
# File: .../stud/nietsch/Studienarbeit/StP_Tycoon/start*
# Original-File: tw*
# Inhalt: Startskript fuer Tycoon in separatem "cmdtool"-Fenster,
#         gueltig auf der dbis11.
# Tycoon-Statement "do exit;" schliesst "cmdtool" automatisch.
#
# alternativer Aufruf:
# --------------------
# cmdtool -I tycoon\ -restart
# Allerdings kann mit diesem Aufruf das Tycoon-Fenster
# nicht durch "do exit;" wieder geschlossen werden.
#
# ======================================================
# WICHTIG: "newgrp" auskommentiert,                     !
#           muss noch beruecksichtigt werden...         !
# ======================================================

#set echo

usage() {
  echo "Usage: start workspace-name";
  exit 99;
}

if [ "$1" = "" ] ; then
  usage
fi

if [ "$2" != "" ] ; then
  usage
fi

if cd /local/teamware/`whoami`/$1; then

OS_VERSION=`uname -r`

export TYCOON_ROOT
export TYCOON_HOST
export CODEMGR_DIR_FLP

case "${OS_VERSION}" in
  4.*) TYCOON_HOST=sunos4 ;;
  5.*) TYCOON_HOST=sunos5 ;;
  *)
    echo "unknown system version: ${OS_VERSION}"
    exit 1
    ;;
esac

cd /local/teamware/`whoami`/$1
TYCOON_ROOT=`pwd`/tycoon
```

```
CODEMGR_DIR_FLP=/local/dbis1/software/etc/tw-share.dir.flp
echo TYCOON_ROOT=$TYCOON_ROOT
echo TYCOON_HOST=$TYCOON_HOST
echo CODEMGR_DIR_FLP=$CODEMGR_DIR_FLP
echo cd `pwd`
# echo newgrp tycoon
# newgrp  tycoon

# fi

echo "cd tycoon/bootlib"
cd tycoon/bootlib
echo "Start Tycoon..."
cmdtool /local/tw1/`whoami`/tycoon/tycoon/bin/sunos5/tycoon -restart

fi
```

A.4 Tycoon-Skript Beispiel1.tyc

```
(* ------------------------------------------------------------------ *)
(* Datei:        Beispiel1.tyc                                        *)
(* Autor:        Oliver Nietsch                                       *)
(*                                                                    *)
(* Universitaet Hamburg                                               *)
(* Fachbereich Informatik                                             *)
(* Arbeitsbereich DBIS                                                *)
(*                                                                    *)
(* Inhalt: Tycoon-Anweisungen zum exemplarischen Aufruf der gesamten  *)
(*         StP Funktionalitaet auf Basis des StP API Datentyps        *)
(*         "oms_node_tp"                                              *)
(*                                                                    *)
(* Voraussetzungen: libstp.so in ${TYCOON_ROOT}/lib/${TYCOON_HOST}    *)
(* Tycoon-Bibliothek: ${TYCOON_ROOT}/bootlib/stpenv                   *)
(*                                                                    *)
(* Version vom  7. 2.1996                                             *)
(* ------------------------------------------------------------------ *)

import omsError;

import omsTime;
import omsTransaction;
import omsRepository;

import omsApplicationType;
import omsObjectId;

import omsIdList;
import omsCollection;
import omsBigCollection;
import omsNode;


let annot = 7;
let projdir = "/local/dbis11/software/StP/project";
let system = "tutor";
let query = "node[name=\'interessanter Node\' && type=701576 && scope_node_id=0
&& sig=\'moin_moin\']";

let name = "interessanter Node";
let type = omsApplicationType.type;
let scope = omsObjectId.optional();
let sig = "moin_moin";


oms_repos_tp Functions:
=======================

omsRepository.init();
                         (* Setzen der StP Umgebungsvariablen: *)
                         (* Sichtbarkeitsbereich: Nur innerhalb des *)
                         (* momentan aktiven Tycoon-Subprozesses *)

let rep = omsRepository.connectSystem(projdir system);

omsRepository.isOpen(rep);
let serverName = omsRepository.currentServerName(rep);
let repName = omsRepository.currentReposName(rep);
```

```
omsRepository.currentReposId(rep);

let rep = omsRepository.connectRep(serverName repName);

omsRepository.disconnect(rep);


oms_node_tp Creation & Retrieval Functions:
===========================================

let node1 = omsNode.create(rep name type omsObjectId.optional() sig);

let node2 = omsNode.find(rep name type omsObjectId.optional() sig);

let node2 = omsNode.findById(rep id);
let attributeRWC = omsNode.name(node2);
  let name = attributeRWC.get();

let node2 = omsNode.findByQuery(rep omsQuery.new("node[name=\'interessanter Node\' && type=701576
&& scope_node_id=0 && sig=\'moin_moin\']"));
  let scopeNodeId = omsNode.id(node4);
    omsObjectId.toInt(scopeNodeId);


oms_node_tp Utility Functions:
==============================

let node4 = omsNode.copy(node2);
let attributeRWC = omsNode.signature(node4);
  let signature = attributeRWC.get();

omsNode.equal(node2 node4);
omsNode.free(node4);
omsNode.printImageAssign("hallo");


oms_node_tp Attribute Access & Assign Functions:
================================================

let attributeR = omsNode.reposId(node2);
  let reposId = attributeR.get();
let id = omsNode.id(node2);
  omsObjectId.toInt(id);
let attributeRWC = omsNode.name(node2);
  let name = attributeRWC.get();
let attributeRW = omsNode.type(node2);
  let type = attributeRW.get();
let attributeRWO = omsNode.scopeNode(node2);
  let scopeNodeId = attributeRWO.get();
    omsObjectId.toInt(scopeNode);
let attributeRWC = omsNode.signature(node2);
  let signature = attributeRWC.get();
let attributeRWO = omsNode.annotFile(node2);
  let annotFileId = attributeRWO.get();
    omsObjectId.toInt(annotFile);

let attributeRWC = omsNode.name(node2);
  let name = attributeRWC.copy();
let attributeRWC = omsNode.signature(node2);
  let signature = attributeRWC.copy();
```

```
let attributeRWC = omsNode.name(node2);
  attributeRWC.set("hallo Node");
let attributeRWC = omsNode.type(node2);
  attributeRWC.set(type);
let attributeRWO = omsNode.scopeNode(node2);
  attributeRWO.set(scopeNode);
let attributeRWC = omsNode.signature(node2);
  attributeRWC.set("moin");
let attributeRWO = omsNode.annotFile(node2);
  attributeRWO.set(annotFile);

omsNode.compareAttrs(node2 name type scope sig);


oms_node_tp Repository Management Functions:
============================================

omsNode.update(node1);
omsNode.delete(node2);
omsNode.check(node1);


oms_node_tp Typed Collection Functions:
=======================================

let coll1 = omsCollection.create(rep :omsRep.Node :omsRep.NodeHandle omsRep.nodeDescriptor query);
  let node10 = coll1.get(0);
    let attributeRWC = omsNode.name(node10);
    let name = attributeRWC.get();

coll1.add(node2);
coll1.addUnique(node2);
coll1.delete(node2);
coll1.deleteEquivalent(node2);
coll1.count();
coll1.free();
coll1.freeAll();
coll1.freeContents();


oms_node_tp Typed Big Collection Functions:
===========================================

let bigcoll1 = omsBigCollection.create(rep :omsRep.Node :omsRep.NodeHandle
omsRep.nodeDescriptor query);
  let node11 = bigcoll1.next();

bigcoll1.free();
bigcoll1.reset();


oms_node_tp Navigation Functions:
=================================

let attributeRWO = omsNode.scopeNode(node2);
  let scopeNode = attributeRWO.deref();
let attributeRWO = omsNode.annotFile(node2);
  let annotFile = attributeRWO.deref();
```

```
let coll2 = omsNode.notes(node2);
let coll3 = omsNode.viewpoints(node2);
let coll4 = omsNode.scopedNodes(node2);
let coll5 = omsNode.scopedLinks(node2);
let coll6 = omsNode.scopedCntxs(node2);
let coll7 = omsNode.outLinks(node2);
let coll8 = omsNode.inLinks(node2);
let coll9 = omsNode.nodeRefs(node2);


oms_time_tp Functions:
======================

omsTime.inputFormatAssign("%C");
omsTime.outputFormatAssign("%C");
omsTime.localEnvironmentAssign(timeText);
let timeText = omsTime.toText(time);
omsTime.textToTime(timeText);
let time = omsTime.now();

omsTime.relTextToTime(timeText);


oms_txn_tp Functions:
=====================

omsTransaction.start(rep);
omsTransaction.inProgress();
omsTransaction.commit(rep);
omsTransaction.abort();
omsTransaction.reset();


oms_id_list Functions:
======================

let idlist = omsIdList.create(rep omsPDMType.node "oliver" query);
idlist.free();
omsIdList.freeAll();


oms_app_type_tp Functions:
==========================

let annotFilename = omsApplicationType.annotFilename(rep omsApplicationType.type);
let appTypeCode = omsApplicationType.codeGet(rep omsPDMType.node 0);
omsApplicationType.codeToText(appTypeCode);

omsApplicationType.comment(appTypeCode);
omsApplicationType.method(appTypeCode "Attribute");

let pdmType = omsApplicationType.pdmType(appTypeCode);

omsApplicationType.printString(appTypeCode);
omsApplicationType.textGet(rep omsPDMType.node 0);
omsApplicationType.textGetCopy(rep omsPDMType.node 0);
let appTypeCode2 = omsApplicationType.textToCode(rep omsPDMType.node "hallo");

let datatype = omsApplicationType.datatype(appTypeCode);
```

```
let filename = omsApplicationType.filename();


oms_pdm_type_tp Functions:
==========================

omsPDMType.toText(pdmType);
let pdmType2 = omsPDMType.fromText("node");


Error Handling:
===============

let errorHandler(isError :Bool num :Int msg :String) :Ok =
  begin
    (* Other statements ... *)
    raise omsError.omsError end
  end;

omsError.setErrorHandler(errorHandler);
```

Literaturverzeichnis

Balzert 93: H. Balzert. *CASE — Systeme und Werkzeuge.* Bibliographisches Institut, Mannheim, 1993.

Geisler 95: A. Geisler. „Basisdienste zur Gestaltung einer reflektiven grafischen Entwicklungsumgebung für eine persistente Programmiersprache". Diplomarbeit, Universität Hamburg, 1995.

IDEStPCust 94: Interactive Development Environments Inc. (Hrsg.). *Software through Pictures Core — Customizing StP*, 1994.

IDEStPOMS 94: Interactive Development Environments Inc. (Hrsg.). *Software through Pictures Core — Object Management System*, 1994.

IDEStPPC 93: Interactive Development Environments Inc. (Hrsg.). *Software through Pictures — Product Catalog 1993*, 1993.

IDEStPQRS 94: Interactive Development Environments Inc. (Hrsg.). *Software through Pictures Core — Query and Reporting System*, 1994.

Klees, Schmauch 94: F. Klees und C. Schmauch. „Navigator: StP/OMT — Software-Entwicklung nach Rumbaugh". In: *iX, Nr. 4, Hannover*, Seite 62ff., 1994.

Mathiske et al. 93: B. Mathiske, F. Matthes, und S. Müßig. „The Tycoon System and Library Manual". DBIS Tycoon Report 212-93, Universität Hamburg, 1993.

Matthes 93: F. Matthes. *Persistente Objektsysteme. Integrierte Datenbankentwicklung und Programmerstellung.* Springer, Berlin, 1993.

Rumbaugh et al. 91: J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, und W. Lorensen. *Object-Oriented Modeling and Design.* Prentice Hall, London, 1991.

Thomas, Nejmeh 92: I. Thomas und B.A. Nejmeh. „Definitions of Tool Integration for Environments". In: *IEEE Software, Nr. 8, New York*, Seite 29–35, 1992.