Universität Hamburg
FB Informatik
Datenbanken und Informationssysteme

Studienarbeit

Hypertextbasierte Visualisierung von *Tycoon*-Bibliotheken in Netzwerken.

April 1996

Kay Ramme
Buchsbaumweg 16
22299 Hamburg
Tel.: 040 / 511 59 61

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

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Ein Programmierer, der sich in ein neues Softwaresystem einarbeiten möchte, hat eine Reihe von Aufgaben vor sich. Er muß sich einerseits mit der Syntax und Semantik der Programmiersprache auseinandersetzen und sich andererseits in die zugehörigen Softwarebibliotheken einarbeiten. Bibliotheken sind ein wichtiger Teil eines Softwaresystems, denn sie erlauben es, Programmcode wiederzuverwenden und abstrakte Problemlösungen auf neue Probleme anzuwenden [Jung95].

Softwarebibliotheken gliedern sich meist in kleinere Einheiten (Module). Diese Einheiten bestehen aus zwei Teilen, einer Schnittstelle (engl. *interface*) und einer Implementation. Die Schnittstelle beschreibt die Menge der Funktionen und Datentypen, die eine Einheit exportiert. Zu einer Schnittstelle kann es auch mehrere Implementationen geben. Ziel dieser Einheitenbildung ist es, logisch zusammenhängenden Programmcode zu implementieren bzw. zu beschreiben. Manche Softwaresysteme erlauben es auch, Bibliotheken rekursiv, d.h. aus anderen Bibliotheken aufzubauen.

Es kann zwischen zwei Arten von Programmierern als Anwender der Bibliothekseinheiten unterschieden werden. Anwendungsprogrammierer sind daran interessiert, die von den Softwarebibliotheken angeboten Funktionen zu verwenden. Während der Einarbeitung werden sie sich mit den Schnittstellenbeschreibungen der Bibliotheken befassen und auseinandersetzen. Die Aufgabe der Systemprogrammierer beinhaltet hingegen die Wartung und Erweiterung der Bibliotheken. Während der Einarbeitung werden sie sich nicht nur mit den Schnittstellenbeschreibungen, sondern auch mit den Implementierungen beschäftigen.

Ein Programmierer, der sich in eine Softwarebibliothek einarbeitet, wird als erstes versuchen, sich einen Überblick zu verschaffen. Der Anwendungsprogrammierer wird die Schnittstellenbeschreibungen der Einheiten der Bibliothek betrachten und versuchen, Funktionen zu finden, mit deren Hilfe er seine Anwendung implementieren kann. Der Systemprogrammier wird sich hingegen auch die Implementierungen der Bibliothekseinheiten ansehen, da seine Aufgabe die Wartung und Erweiterung der Bibliothek beinhaltet.

Da Bibliothekseinheiten auf anderen Bibliotheken basieren können, wird der Programmierer bei der Betrachtung einer Schnittstellenbeschreibung oder einer Implementation versuchen zu ermitteln, welche Funktion die aus anderen Bibliothekseinheiten importierten Objekte haben. Er wird also den Beziehungen innerhalb der Bibliothekseinheiten folgen und sich die jeweiligen Schnittstellenbeschreibungen oder Implementierungen ansehen.

Der Aufwand zur Einarbeitung in eine Softwarebibliothek ist u.a. abhängig von der Anzahl ihrer Module und Schnittstellen und der Menge der Beziehungen, die zwischen diesen bestehen. Die Standardbibliothek des *Tycoon*-Systems beinhaltet beispielsweis 24 Module und 24 Schnittstellen. Bei einem geschätzten Import von 150 Programmobjekten ergibt sich eine Menge von 7200 Beziehungen.

Ziel dieser Studienarbeit ist es, ein Werkzeug zu entwickeln, mit dessen Hilfe die Einarbeitung in die Softwarebibliotheken des *Tycoon*[1]-Systems erleichtert wird. Das Werkzeug soll es erlauben, die Bezüge der Bibliothekseinheiten zueinander aufzulösen und zu visualisieren. Die Bibliothekseinheiten werden im weiteren Module genannt.

Das Projekt gliedert sich in folgende Teile:

- Ermittlung der darzustellenden Beziehungen der Module.
- Visualisierung der Beziehungen, d.h. Wahl der Darstellungsart und Wahl der Darstellungsmittel.

1.2 Beziehungen

Die Objekte einer Programmbeschreibung stehen miteinander in einer Vielzahl von Beziehungen. Eine Art von Beziehung ist die referentielle. Es wird beispielsweise am Anfang des Programms eine Variable mit einem Bezeichner assoziiert (Definition, Deklaration), die im weiteren Programm mit Hilfe des Bezeichners referenziert wird. Viele Programmiersprachen erlauben kompliziertere referentielle Beziehungen als die direkte. Z.B. ist es möglich, Namensräume (engl. *scopes*) zu bilden und ineinander zu schachteln. Referenzen werden also nicht nur durch die Gleichheit von Bezeichnern aufgelöst, es müssen auch die zugehörigen Namensräume in Beziehung stehen, wobei Objekte übergeordneter Namensräume von lokalen Objekten verdeckt werden können (siehe Abb. 1). Die Auflösung der Referenzen ist also abhängig von der Sichtbarkeit der Objekte.

Programme und Bibliotheken bzw. Module werden meist unter Verwendung anderer Bibliotheken erstellt. Die Objekte der Programme, Biliotheken und Module stehen also auch in Beziehung zueinander. Ein weiteres Beispiel für die Sichtbarkeit von Objekten ist ein Modul, welches zwei Funktionen beinhaltet, wobei es eine exportiert und die andere nur lokal verfügbar ist (siehe Abb. 2).

Die Art dieser Zusammenhänge ist Teil der Semantik, wie sie im Übersetzer

[1] Tycoon: TYped COmmunicating Objects in open eNviroments.
Das *Tycoon*-System wurde am Arbeitsbereich Datenbanken und Informationssysteme des Fachbereichs Informatik der Universität Hamburg entwickelt.

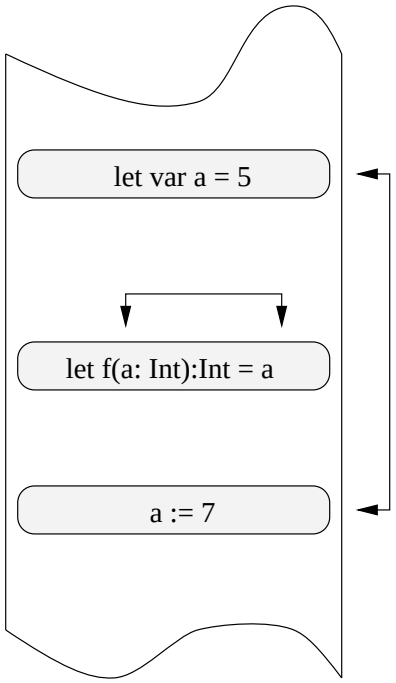

Abbildung 1: Referentielle Beziehungen

bzw. der Spezifikation des Übersetzers definiert ist. Die meisten Übersetzer lösen während des Übersetzens referentielle Zusammenhänge bezüglich Variablen, Konstanten und Funktionen auf (*static scoping*).

Es sind aber auch andere Arten von Beziehungen zwischen Objekten des Programmcodes möglich. So bietet z.B. die Sprache *TL*[2] des *Tycoon*-Projektes das Konzept des *Subtypisierung*. Typdefinitionen stehen hier nicht in einem referentiellen Verhältnis, sondern in einem strukturellen. Da diese Arten von Beziehungen meist nicht explizit vom Übersetzer aufgelöst werden, ist ihre Ermittlung mit einem größeren Aufwand verbunden. Beispielsweise müßte in *Tycoon* prinzipiell jeder neu definierte Typ mit allen bereits definierten verglichen werden, um festzustellen, mit welchem er in welcher Beziehung steht (Subtyp, Supertyp, kein Bezug).

1.3 Bezugsinformationen

Objekte von Programmbeschreibungen stehen miteinander in Beziehungen. Beziehungen können sich über mehrere oder über nur eine Programmbeschreibung erstrecken. Die Information, die diese Beziehungen beschreibt, ist die Bezugsinformation. Da Bezugsinformationen beliebige Beziehungen der Objekte des Pro-

[2]TL: Tycoon Language.

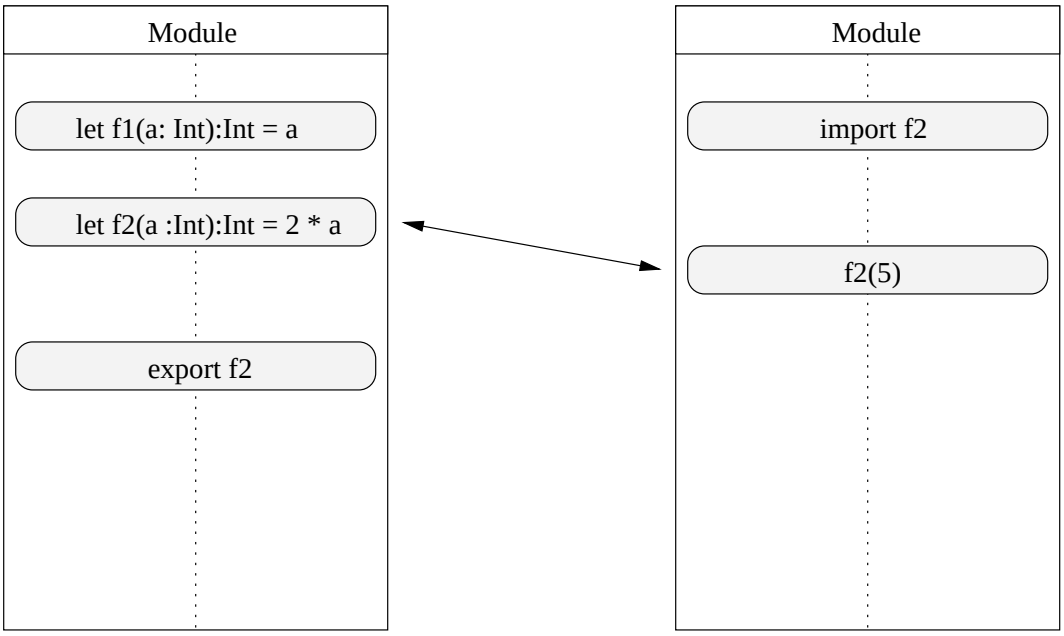

Abbildung 2: Referentielle Beziehungen über Module

grammtextes beschreiben können sollen, bietet sich ihre Darstellung als Relation im mathematischen Sinne an. Aufgabe einer solchen Relation ist es, Teile eines Programmtextes mit Teilen eines anderen Programmtextes in Beziehung zu setzen. Die Menge, über die diese Relation gebildet wird, besteht aus den Textpositionen einer Programmbeschreibung (und der von ihr verwendeten Module). Die Bezugsinformationen innerhalb von und zwischen Programmen lassen sich somit als Textpositionsrelationen darstellen.

2 Entwurf des Visualisierungswerkzeugs

Computerbasierte Visualisierungen basieren auf einer Menge von Daten und einem Programm, das diese Daten visualisiert. Die Daten dieses Projektes sind die Modul-, Schnittstellen- und Bibliotheksbeschreibungen des *Tycoon*-Systems. Die Beziehungen der Objekte der Beschreibungen zueinander ergeben sich aus der Semantik der den Beschreibungen zugrundeliegenden Programmiersprache *TL*. Thema dieser Studienarbeit ist ein Programm zur Visualiserung der Modul-, Schnittstellen- und Bibliotheksbeschreibungen sowie der Beziehungen der Objekte der Beschreibungen.

In den nächsten Abschnitten werden Ort sowie Zeitpunkt der Erzeugung der Bezugsinformationen bestimmt. Es werden Darstellungsart und Darstellungsmittel ausgewählt. Die Struktur des Werkzeugs, die bestimmt ist durch die Darstellungsart und die Art der Erzeugung der Bezugsinformationen, wird ermittelt.

2.1 Erzeugung der Bezugsinformationen

Um die Beziehungen der Objekte einer Programmbeschreibung zueinander darzustellen, werden Bezugsinformationen benötigt (vgl. Kapitel 1.3). Diese Bezugsinformationen sind durch die Semantik der der Programmbeschreibung zugrundeliegenden Programmiersprache festgelegt. Sie sind also implizit in der Programmbeschreibung enthalten. Für die Visualisierung werden die Bezugsinformationen jedoch explizit benötigt.

Hierdurch ergeben sich folgende Fragen:

- Wo sollen die Bezugsinformationen erzeugt werden, d.h. welches Programm erzeugt die Informationen?
- Wann sollen die Bezugsinformationen erzeugt werden, bei Bedarf oder schon vorher?

Wenn die Bezugsinformationen vor dem Bedarf erzeugt werden, müssen sie gespeichert werden.

Es gibt folgende Möglichkeiten der Speicherung:

- Die Informationen werden alle zusammen, d.h. zentral gespeichert.
- Die Informationen werden verteilt gespeichert.

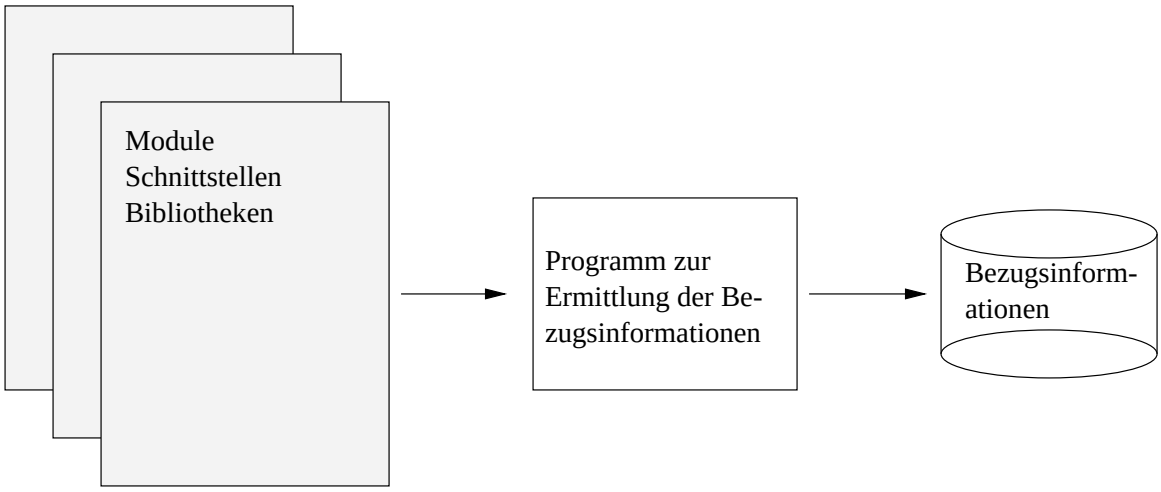

Abbildung 3: Erzeugung der Bezugsinformationen

2.1.1 Erzeugungsort (Programm)

Prinzipiell gibt es zwei Möglichkeiten, die Bezugsinformationen zu ermitteln. Einerseits kann ein separates Programm, eine Art Übersetzer, unter Verwendung der Semantik der verwendeten Programmiersprache, die Bezugsinformationen direkt erzeugen (vgl. Abb. 3). Andererseits kann der Übersetzer die Bezugsinformationen während des Übersetzens, also als Seiteneffekt, generieren.

Die Erzeugung der Bezugsinformationen in einem externen Programm erfordert einen Parser für die Programmiersprache, sowie die algorithmische Umsetzung jener Teile der Semantik, die die zur Darstellung ausgewählten Beziehungen beschreiben. Für die meisten Programmiersprachen gibt es keine formale Beschreibung der Semantik ihrer Programmbeschreibungen. Eine solche formale Semantik ist jedoch notwendig, damit verschiedene Übersetzer einer Programmbeschreibung zu gleichen Ergebnissen kommen. Ein Ergebnis wäre hier die Menge von Bezugsinformationen.

Fast alle für die Erzeugung von Bezugsinformationen notwendigen Mechanismen sind bereits Teil des Übersetzers. Das Erstellen eines eigenen Programms zur Erzeugung von Bezugsinformationen wäre also doppelte Arbeit.

Da die Semantik der Programmiersprache bereits innerhalb des Übersetzers implementiert ist, kann durch eine Ergänzung bzw. Modifikation des Übersetzers dieser veranlaßt werden, die Bezugsinformationen während des Übersetzens zu erzeugen. Um den Aufwand gering zu halten, wird in diesem Projekt dieser Ansatz verfolgt.

2.1.2 Erzeugungszeitpunkt

Die Bezugsinformationen können jeweils bei Bedarf oder einmalig zur Übersetzungszeit erzeugt werden. Werden die Informationen zur Übersetzungszeit erzeugt, so belegen sie anschließend Speicherplatz, sind jedoch bei Bedarf sofort verfügbar.

Der benötigte Speicherplatz ergibt sich aus der Größe der Beschreibung einer Beziehung und der Anzahl der Beziehungen. Die Beschreibung einer Beziehung beinhaltet die Positionen der in Beziehung stehenden Programmbeschreibungen, sowie die Positionen der Bezug aufeinander nehmenden Objekte der Programmbeschreibungen.

In einem hierachischen Dateisystem und bei einer durchschnittlichen Pfadlänge von 40 Zeichen ergibt sich für den benötigten Speicherplatz einer Beziehungsbeschreibung eine Menge von ca. 80 Zeichen. Bei 100 Beziehungen pro Bibliothekseinheit (Inter- und Intrabibliothekseinheitenbeziehungen) ergibt sich pro Bibliothekseinheit ein benötigter Speicherplatz von 8000 Zeichen. Bei einer Menge von 50 Bibliothekseinheiten pro Bibliothek sind das 20000 Zeichen je Bibliothek.

Die Erzeugung der Bezugsinformationen bei Bedarf kann, je nach Art der Beziehungen, viel Rechenzeit benötigen. Besonders bei nicht referentiellen Beziehungen (vgl. Kapitel 1.2) kann die nötige Rechenzeit sehr groß werden, da der Aufwand zu ihrer Bestimmung exponentiell wachsen kann.

Da die Visualisierung der Bibliothekseinheiten dynamisch erfolgen soll, wird ein schneller Zugriff auf die Bezugsinformationen benötigt. Ein schneller Zugriff ist nur bei Vorhaltung möglich. Aus diesem Grund werden die Bezugsinformationen innerhalb dieses Projektes einmalig zur Übersetzungszeit erzeugt.

2.1.3 Persistente Speicherung der Bezugsinformationen

Die Bezugsinformationen einer Softwarebibliothek können an unterschiedlichen Stellen gespeichert werden. Zum einen ist es möglich, alle Informationen an einem Ort zu halten (zentral). Bei einer Anfrage ist nur dieser eine Ort zu durchsuchen. Zum anderen können die Bezugsinformationen derart verteilt werden, daß sie den jeweiligen Einheiten der Softwarebibliothek zugeordnet werden (dezentral). Diese Lösung hat den Vorteil, daß beispielsweise bei einer Neuübersetzung eines Moduls nicht die gesamten gespeicherten Bezugsinformationen bezüglich des Softwaresystems durchgesehen werden müssen. Die Bezugsinformationen für Schnittstellen, Bibliotheken und Module lassen sich weitgehend unabhängig voneinander verwalten. Diesem Vorteil steht allerdings der Nachteil des höheren Verwaltungsaufwandes gegenüber.

Dieser höhere Verwaltungsaufwand resultiert daraus, daß es bei einer verteilten Speicherung für jede Bibliothekseinheit eine Menge von Bezugsinformationen gibt. Da jedoch auch Beziehungen zwischen den Bibliothekseinheiten bestehen (vgl. Kapitel 1.2), bestehen auch Beziehungen zwischen den Bezugsmengen der Bibliothekseinheiten. Diese Beziehungen müss en nun auch verwaltet werden.

Die Bezugsinformationen werden innerhalb dieser Studienarbeit dezentral Verwaltet. Diese geschieht auch im Hinblick auf eine Integration der Verwaltung der *TL*-Programmbeschreibungen in das *Tycoon*-System.

2.2 Art der Darstellung

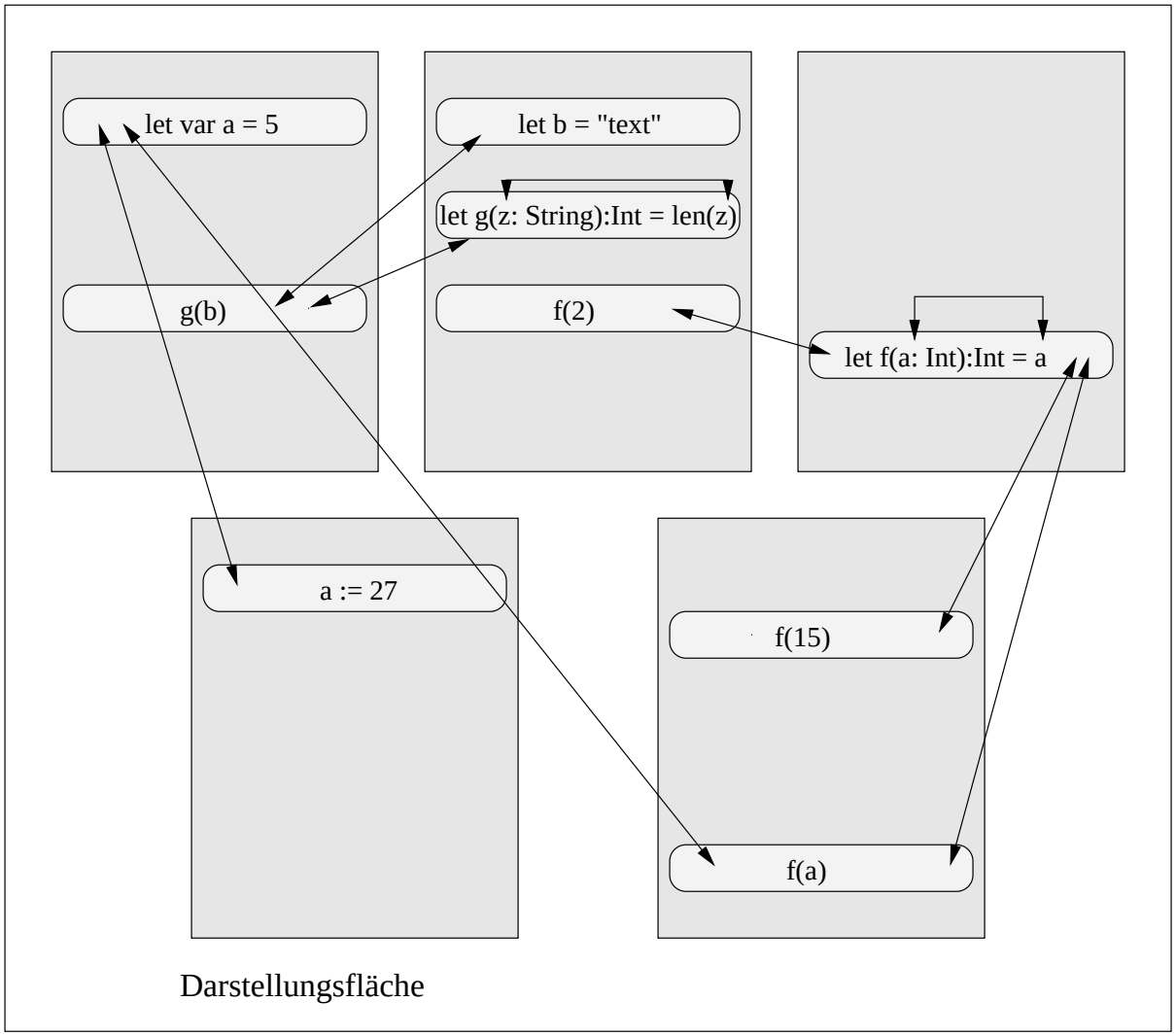

Abbildung 4: Darstellung aller Beziehungen

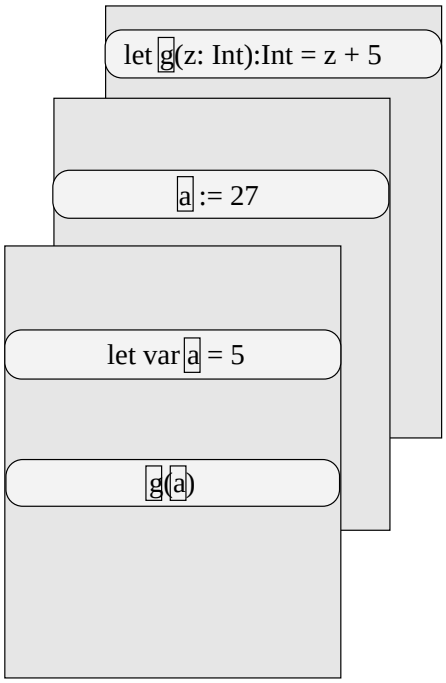

Abbildung 5: Hypertext mir Markierungen

Für die Visualisierung von miteinander in Beziehung stehenden Programmtexten und Bibliotheken ist eine Art der Darstellung nötig, die es erlaubt, einen Text mit Bezügen auf andere Texte anzuzeigen.

Es gibt verschiedene Möglichkeiten, Beziehungen zwischen Texten visuell darzustellen. Beispielsweise können alle Texte, die miteinander in Beziehung stehen, gleichzeitig dargestellt werden, wobei die Beziehungen zwischen Objekten der Programmbeschreibungen, z.B. mit Hilfe von Pfeilen ausgedrückt werden können (siehe Abb. 4). Bei einer großen Anzahl von zusammenhängenden Texten wird eine solche Darstellung jedoch schnell unübersichtlich.

Eine andere Möglichkeit der Darstellung von Bezugsinformationen zwischen Texten ist der Hypertext. Hierbei wird jeweils nur ein Text (bzw. ein Ausschnitt) angezeigt. Die Textstellen des Textes, die mit anderen Texten in Beziehung stehen, werden dabei markiert. Durch Auswählen einer so markierten Textstelle wird der gerade sichtbare Text durch jenen ersetzt, mit dem die ausgewählte Textstelle in Bezug steht. Mit Hilfe des Hypertextes kann auch eine große Anzahl von zusammenhängenden Texten dargestellt werden.

Aus Gründen der Übersichtlichkeit wurde der Hypertext als Darstellungsart gewählt.

2.3 Struktur des Werkzeugs

Bei der Wahl von Hypertext als Darstellungsart und dem Übersetzer als Erzeuger der Bezugsinformationen, ergibt sich die in Abbildung 6 gezeigte Struktur des entwickelten Werkzeugs.

Es gibt zwei Datenmengen, die Menge der Programmbeschreibungen (Modul-, Schnittstellen- und Biliotheksbeschreibungen) sowie die Menge der Bezugsinformationen. Die Bezugsinformationen werden aus den Programmbeschreibungen vom Übersetzer extrahiert. Aus den Programmbeschreibungen wird mit Hilfe der Bezugsinformationen eine Darstellungsbeschreibung für Hypertext erzeugt. Die Darstellungsbeschreibung wird vom Darstellungsmittel als Hypertext dargestellt.

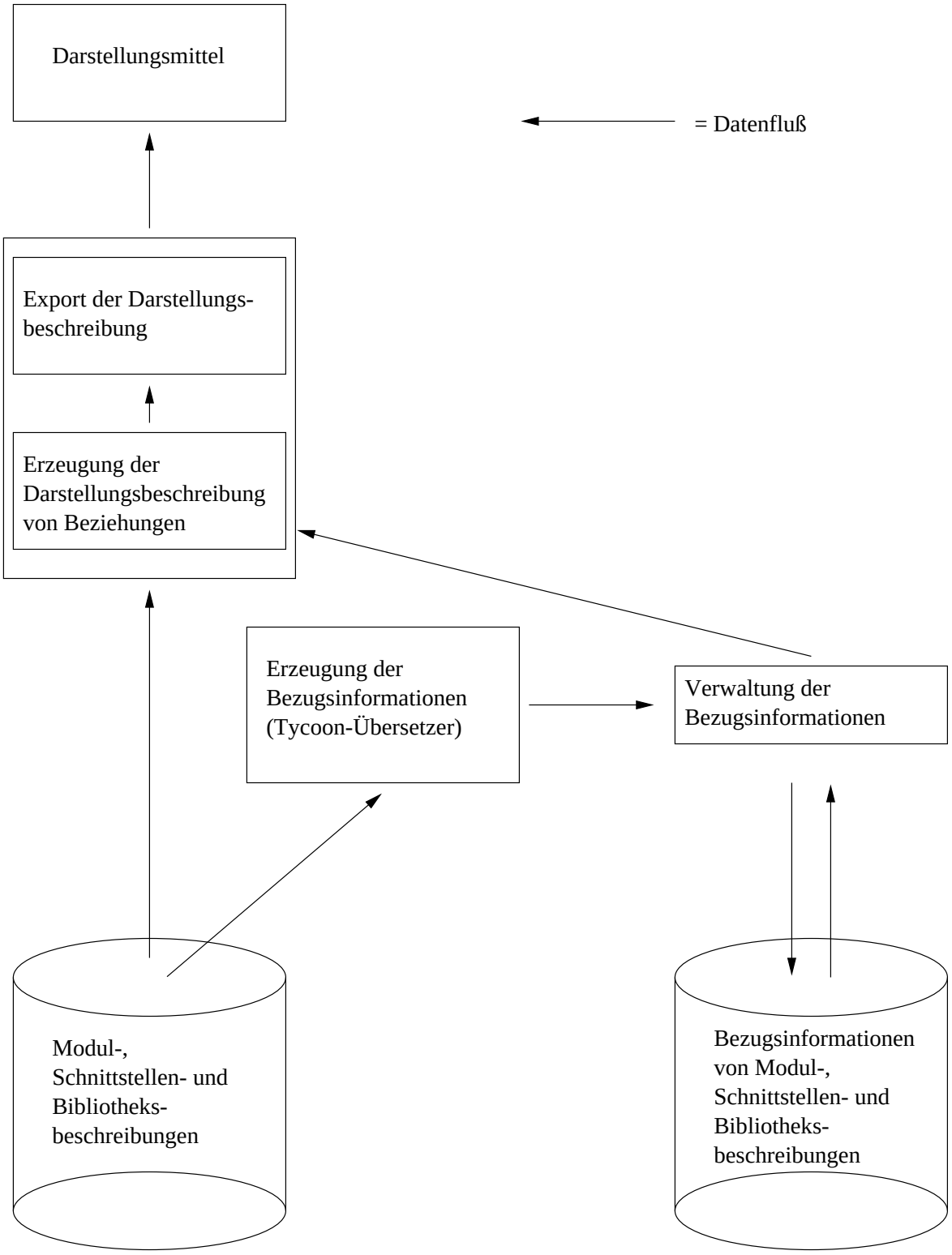

Abbildung 6: Aufbau des Visualisierungswerkzeugs

3 Visualisierung

Es gibt verschiedene Möglichkeiten miteinander in Beziehung stehende Texte darzustellen. Für die Visualisierung von Programmtexten im Rahmen dieser Studienarbeit wird Hypertext verwendet (vgl. Kapitel 2.2) .

In diesem Kapitel werden die Anforderungen an eine benötigte Visualisierungstechnologie ermittelt, sowie die grundlegenden Mechanismen der ausgewählten Visualisierungsplattform (WWW) erläutert.

3.1 Anforderungen und Auswahl

Folgende Anforderungen werden an die Visualisierungstechnologie gestellt:

- Die Visualisierungstechnologie soll es erlauben, Programmtexte innerhalb eines Netzwerkes verfügbar zu machen, d.h. die Programmtexte sollen zentral gespeichert werden und von jeder Station des Netzwerkes einsehbar sein. Sie soll eine einfache Navigation innerhalb der Programmtexte bzw. Bibliotheken ermöglichen.
- Der Aufwand zur Darstellung der Programmtexte und ihrer Bezüge soll gering gehalten werden, d.h. die Visualiserungstechnologie soll die Darstellung von Programmtexten und deren Bezügen auf möglichst hohem Niveau unterstützen.

Verschiedenen Visualisierungstechnologien unterstützen die Darstellung von Hypertext auf unterschiedlichem Niveau. Mit Hilfe der *StarView*-Bibliothek ließe sich z.B. ein betriebssystemunabhängiges Programm schreiben, das Hypertext darstellt und es erlaubt, darin zu navigieren. Nachteil einer solchen Lösung ist der Programmieraufwand. Außerdem ist die Portierung auf solche Systeme eingeschränkt, für die die ausgewählte Bibliothek verfügbar ist. Auch der Umfang einer ausgewählten Bibliothek spielt eine Rolle. Die *StarView*-Bibliothek beinhaltet beispielsweise keine Netzwerkunterstützung.

Das *Info*-Programm [FOX89(FSF/GNU)] erlaubt es, Hypertext anzuzeigen, indem die in Beziehung stehenden Texte in einem speziellen Format gespeichert werden. Dieses Programm arbeitet jedoch nicht netzwerktransparent. Die darzustellenden Texte müssen vor Aufruf des Programms bereitstehen, sie können nicht erst bei Bedarf erzeugt werden. Der Zeitpunkt ihrer Erzeugung ist also festlegt.

Das WWW (World Wide Web) erlaubt es, Hypertext zentral zu speichern und in einem Netzwerk zugreifbar zu machen. Ferner kann der anzuzeigende Text auch erst bei Bedarf erzeugt werden. Dies gestattet es, den Erzeugungszeitpunkt

unabhängig von der Visualisierungstechnologie zu wählen. Die an die Visualiseriungstechnologie gestellten Anforderungen werden von den Mechanismen des WWW erfüllt.

3.2 Darstellung von Hypertext im World Wide Web

Im WWW sind die meisten Internetdienste, wie z.B. FTP, Gopher und Telnet, zusammengefaßt. Die Einheit, die zur Darstellung von Informationen im WWW verwendet wird, ist die Seite. Eine Seite ist die grafische Darstellung von Informationen, bestehend aus Text, Bildern und Verweisen.

Das WWW basiert auf dem Client/ Server Konzept. Der Client übernimmt die Darstellung einer Seite und stellt dem Benutzer Möglichkeiten zur Navigation zur Verfügung. Die Aufgabe des Servers ist es, eine Menge von gespeicherten Informationen im Netzwerk verfügbar zu machen. Die meisten Clients können auf Server unterschiedlichen Typs zugreifen (FTP, Gopher). Abbildung 8 zeigt die Kommunikation eines Clients mit einem HTTP-Server.

Der Client hat die Auswahl zwischen einer Menge von HTTP-Servern. Die Auswahl eines Servers erfolgt über die zugehörige Adresse. Diese Adresse ist Teil einer URL (siehe Abschnitt 3.4). Je nach Art der Anfrage des Clients gibt der HTTP-Server die angeforderten Daten entweder direkt, aus der Menge der ihm zur verfügung stehenden Daten, zurück oder aktiviert ein Programm zur Erzeugung der Daten (Gateway).

Die meisten Clients sind in der Lage, eine Menge unterschiedlicher Daten anzuzeigen. Die Seite ist jedoch die Basis der Darstellung, da sie die anderen Daten integriert.

Für die Beschreibung von WWW-Seiten gibt es eine Seitenbeschreibungssprache. Diese und die von ihr verwendeten Mechanismen werden im weiteren kurz beschrieben. Außerdem wird darauf eingegangen, wie der HTTP-Server mit den ihm zur Verfügung stehenden Daten umgeht.

3.3 Seitenbeschreibungssprache (HTML)

Seiten des WWW werden mit der Seitenbeschreibungssprache HTML (HyperText Markup Language) beschrieben und können unter anderem Bilder und Verweise auf andere Seiten enthalten. HTML an sich beschreibt Fließtext mit eingestreuten Gliederungsanweisungen.

HTML-Seiten sind hierarchisch aufgebaut. Sie besitzen auf oberster Ebene einen Kopf- (engl. *head*) und einen Rumpfbreich (engl. *body*). Der Kopf beinhaltet den

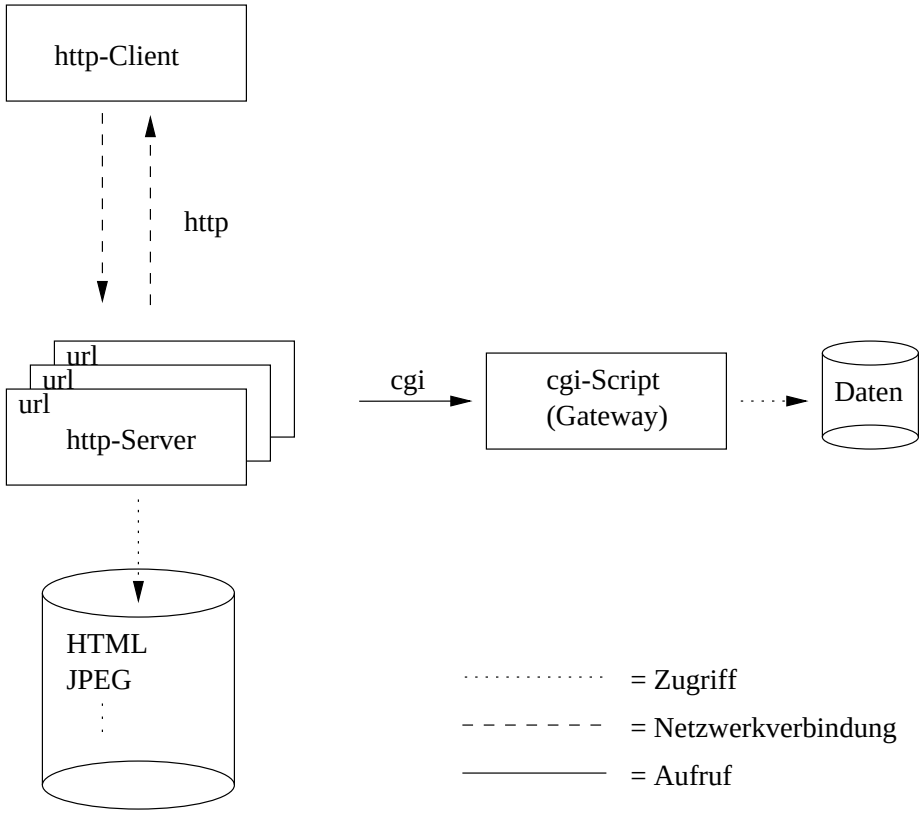

Abbildung 7: Verbindung eines HTTP-Clients mit einem HTTP-Server

Titel, während im Rumpf die eigentliche Seite beschrieben wird.

HTML unterstützt geschachtelte Kapitel, verschiedene Arten von Listen und bietet die Möglichkeit, Grafiken einzubinden. Textteile können derart markiert werden, daß sie kursiv oder fett oder auf andere Weise betont ausgegeben werden. Es ist möglich, Abschnitte vorzuformatieren, so daß sie bei der Ausgabe ihre Formatierung beibehalten. Außerdem können Textstellen als Verweise markiert werden, bei deren Auswahl die aktuelle Seite durch die referenzierte ersetzt wird.

Es folgt ein Beispiel für eine mit HTML beschriebene Seite.

```
<HTML>
 <HEAD>
  <TITLE>home page</TITLE>
 </HEAD>
 <BODY>
  <H1> home page</H1>
  <P>
  Die Quelldateien:
```

```
  <ul>
   <li>
     <A HREF="http://dbis9/cgi-bin/con?FILE+refLib/refLib.tl">refLib.ti</A>

   <li>
     <A HREF="http://dbis9/cgi-bin/con?FILE+refLib/RefDB.ti">RefDB.ti</A>
     :
     <A HREF="http://dbis9/cgi-bin/con?FILE+refLib/refDB.tm">refDB.tm</A>

   <li>
     <A HREF="http://dbis9/cgi-bin/con?FILE+Lines.ti">Lines.ti</A>
     :
     <A HREF="http://dbis9/cgi-bin/con?FILE+lines.tm">lines.tm</A>

  </ul>
 </BODY>
</HTML>
```

Diese Seite stellt einen Auszug einer Liste aller zu diesem Projekt gehörenden *Tycoon*-Schnittstellen, -Module und -Bibliotheken dar. Sie hat einen Kopf und einen Rumpf. Der Kopf beinhaltet den Titel, im Rumpf befindet sich eine Beschreibung einer Liste von Referenzen.

3.4 Seitenreferenzen (URLs)

WWW-Seiten können Verweise, auf andere WWW-Seiten oder andere Daten im Netzwerk, beinhalten. Diese Verweise werden als URLs (Uniform Resource Locator) formuliert. Ein URL ist eine Beschreibung der Position und des Typs eines Dokumentes oder eines Dienstes im Internet. Durch die URLs ist möglich, viele Internetdienste zusammenzufassen.

URLs haben folgenden Aufbau:

```
<Typ des Service>://<Adresse des Servers>[:<Port>]/<WWW-Seite>[?<Parameter>]
```

Der Typ des Service wählt die Art des Protokolls aus. Durch die Adresse wird im Netzwerk ein Computer ausgewählt. Durch den Port wird ein Serviceprogramm auf dem adressierten Computer ausgewählt. Wird kein Port angegeben, wird der Standardport für das ausgewählte Protokoll verwendet. Der optionale Parameter wird nur bei der Protokollart HTTP (siehe Abschnitt 3.5) verwendet.

Es folgt ein Beispiel für einen URL:

Name	Bedeutung
SERVER_SOFTWARE	Name und Version der Serversoftware
SERVER_NAME	Name des Servers
GATEWAY_INTERFACE	Name und Version der Schnittstelle
SERVER_PROTOCOL	Name und Version des Protokolls
SERVER_PORT	Nummer des Ports
REQUEST_METHODE	Art der Parameterübergabe
HTTP_ACCEPT	Datentypen, die der Client verarbeiten kann
SCRIPT_NAME	Name des Gateways
QUERY_STRING	Gateway Paramter
REMOTE_HOST	Name des Clienthosts
REMOTE_ADDR	Adresse des Clienthosts
REMOTE_USER	Name des Benutzers
AUTH_TYPE	Authentifizierungsmethode
CONTENT_TYPE	Art der Parameter
CONTENT_LENGTH	Länge der Parameter

Tabelle 1: Umgebungsvariablen eines Gateways

```
http://www.informatik.uni-hamburg.de/entry.html
```

Im HTML-Beispiel in Abschnitt 3.3 sind weitere URLs zu sehen.

3.5 Transferprotokoll (HTTP)

Das HTTP (HyperText Transfer Protocol) beschreibt die Kommunikation des WWW-Clients mit dem WWW-Server. HTTP ist textbasiert, d.h. Anfragen und Anfrageergebnisse werden als lesbarer Text transportiert. Für eine Anfrage eines Clients an einen Server wird jeweils eine neue Verbindung aufgebaut, die nach Übertragung der WWW-Seite wieder abgebaut wird (siehe Abb. 8), HTTP ist also in dem Sinne verbindungslos, als daß eine Verbindung vom Client zum Server nur während der Übertragung von Daten besteht. Nach Verbindungsaufbau überträgt der Client Informationen über sich (Software, Version), den Namen der angeforderten Seite sowie eventuell einen Parameterstring. Dieser Paramterstring ist Teil des die Daten referenzierenden URLs und wird als Klartext angegeben. Wird das angeforderte Dokument von einem Gateway (siehe Abschnitt 3.6), d.h. dynamisch, erzeugt, wird der Parameterstring dem Gateway als Paramter übergeben. Der Client übermittelt dem Server auch eine Menge mit Dateiformaten, die er anzeigen kann.

3.6 Servererweiterung (CGI)

HTTP-Server besitzen zur Ergänzung ihrer Funktionalität eine Schnittstelle zu externen Programmen, normalerweise das CGI (Common Gateway Interface). Die Programme, die auf dieser Schnittstelle aufsetzen, werden als Gateways bezeichnet, da sie meist Verbindungen zu Datenbanken oder anderen Servern aufbauen. Mit Hilfe des CGI ist es einem HTTP-Client möglich, Parameter an das Gateway zu übergeben. Das CGI spezifiziert weiterhin eine Menge von Umgebungsvariablen, die vom HTTP-Server an das Gateway übergeben werden (siehe Tabelle 1). Abbildung 8 zeigt die Kommunikation zwischen Gateway, HTTP-Client und HTTP-Server.

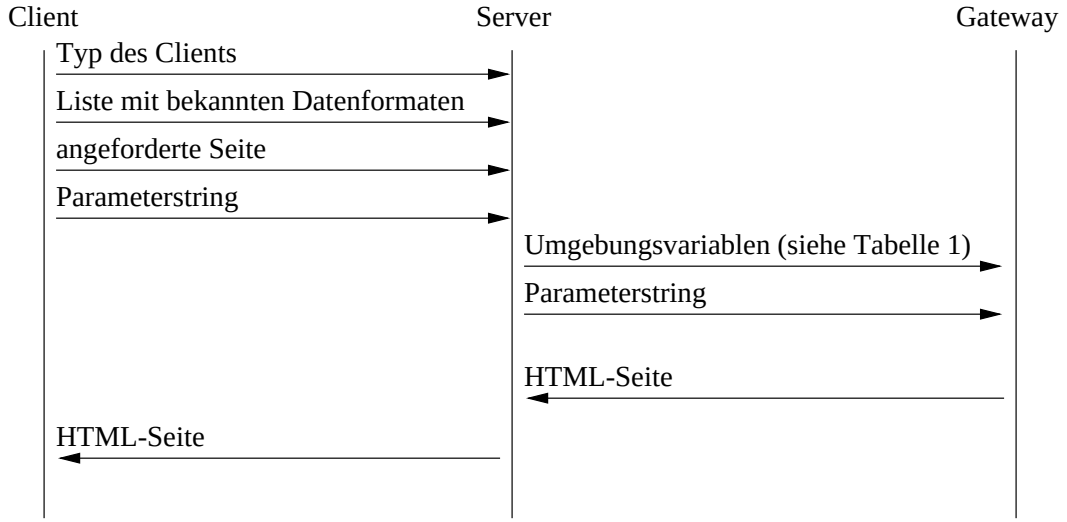

Abbildung 8: Kommunikation zwischen Client, Server und Gateway

4 Protokoll zum externen Aufruf von *Tycoon*-Funktionen (TIP)

Zur Visualisierung von Hypertext soll die Technologie des WWW benutzt werden. Der Hypertext soll jedoch von einem Programm erzeugt werden, das mit Hilfe der Programmiersprache *TL* beschrieben wird. Die aus *TL* generierten Programme befinden sich normalerweise in einem *Tycoon*-Objektspeicher und sind auch nur von hier aus aufrufbar. Benötigt wird also eine Verbindung zwischen WWW und *Tycoon*-Objektspeicher. Diese Verbindung soll es erlauben, Funktionen eines *Tycoon*-Objektspeichers von außerhalb des Objektspeichers, also von Programmen, die sich nicht im Objektspeicher befinden, aufzurufen. Ferner soll es möglich sein, den Funktionen des Objektspeichers beim Aufruf Parameter zu übergeben. Das Ergebnis der Ausführung einer Funktion soll an den Aufrufer zurückgegeben werden.

Da für das *Tycoon*-System bereits eine Bibliothek mit Funktionen zur Kommunikation über ein Netzwerk vorhanden ist, bietet es sich an, eine Verbindung zwischen Programmen innerhalb und außerhalb eines *Tycoon*-Objektspeichers als Client/Server Paar zu implementieren.

Das Protokoll, das die Kommunikation zwischen Client und Server regelt, heißt TIP (*Tycoon Interface Protocol*). Mit TIP ist es einem, sich nicht in einem *Tycoon*-Objektspeicher befindlichen, Programm möglich, in einem *Tycoon*-Objektspeicher befindliche Funktionen aufzurufen. Der Aufrufer kann der Funktion Parameter übergeben. Das Ergebnis der Funktion wird an den Aufrufer zurückgegeben (vgl. Abb. 9).

Es folgt die Beschreibung des Protokolls von Client und Server sowie die Beschreibung eines für diese Studienarbeit implementierten TIP-Clients und eines TIP-Servers.

4.1 Das Protokoll

Das TI-Protokoll ist textbasiert und transferiert ausschließlich Zeichenketten zwischen Client und Server. Nachdem der Client eine Verbindung zu einem TIP-Server aufgebaut hat, schickt er diesem die TIP-Kennung "TIP", ergänzt um die Versionsnummer, und anschließend eine mit einem leeren Paar abgeschlossene Folge von Namen/Werte Paaren. Anhand des ausgezeichneten Wertepaares mit Namen *SERVICE_NAME*, ermittelt der TIP-Server die zu aktivierende Funktion und ruft diese auf. Als Ergebnis dieser Funktion wird vom TIP-Server eine Zeichenkette erwartet, welche an den Client zurückgegeben wird. Anschließend baut der Server die Verbindung zum Client ab. Die Verpflichtung des Clients zur Spe-

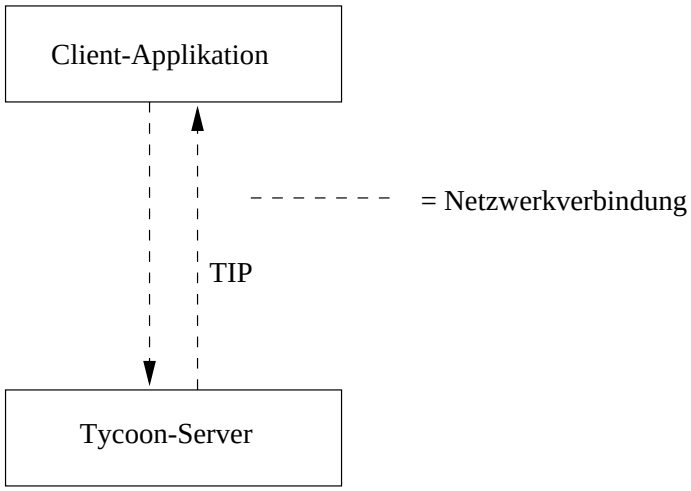

Abbildung 9: Kommunikation externer Programme mit einem *Tycoon*-Server

zifikation des zu erbringenden Service erlaubt es dem TIP-Server, eine Vielzahl von Servicefunktionen zum Aufruf anzubieten.

Abbildung 10 zeigt eine Beispielkommunikation zwischen TIP-Client und TIP-Server. Der TIP-Client wurde in diesem Beispiel von einem CGI-Gateway gestartet. Das Gateway übergibt dem Client seine Umgebungsvariablen sowie den Namen der aufzurufenden Funktion und einen Parameter. Ergebnis ist der in HTML konvertierte Quelltext der *Tycoon*-Schnittstelle `Lines.ti` (im Protokoll abgekürzt dargestellt).

TIP hat nur zwei Kommunikationsphasen, zuerst transferiert der Client die für den Funktionsaufruf nötigen Parameter, anschließend gibt der Server das Ergebnis des Aufrufs an den Client zurück. Es ist auch möglich, ein mehrphasiges Protokoll zu verwenden. Beispielsweise braucht der Client zu Anfang nur den Namen der aufzurufenden Funktion anzugeben. Die Servicefunktion stellt nun für jeden Parameter eine Anfrage an den Server und dieser gibt die Anfrage an den Client weiter. Das Ergebnis wird vom Client über den Server an die Servicefunktion zurückgegeben. Vorteil eines solchen Protokolls ist die erhöhte Flexibilität, da die Servicefunktion je nach Bedarf Anfragen stellen kann. Nachteil ist der erhöhte Programmieraufwand der Servicefunktion. Es ist auch möglich, den TIP-Ansatz sowie den eben beschriebenen zu kombinieren. Dies würde eine Erweiterung des TI-Protokolls sowie der Serverobjektspezifikation (siehe Abschnitt 4.2) erfordern.

4.2 Der Server

Zur Unterstützung der Implementation eines TIP-Servers existiert das Tycoon-Modul *TipServer*. Abbildung 11 zeigt die exportierten Funktionen.

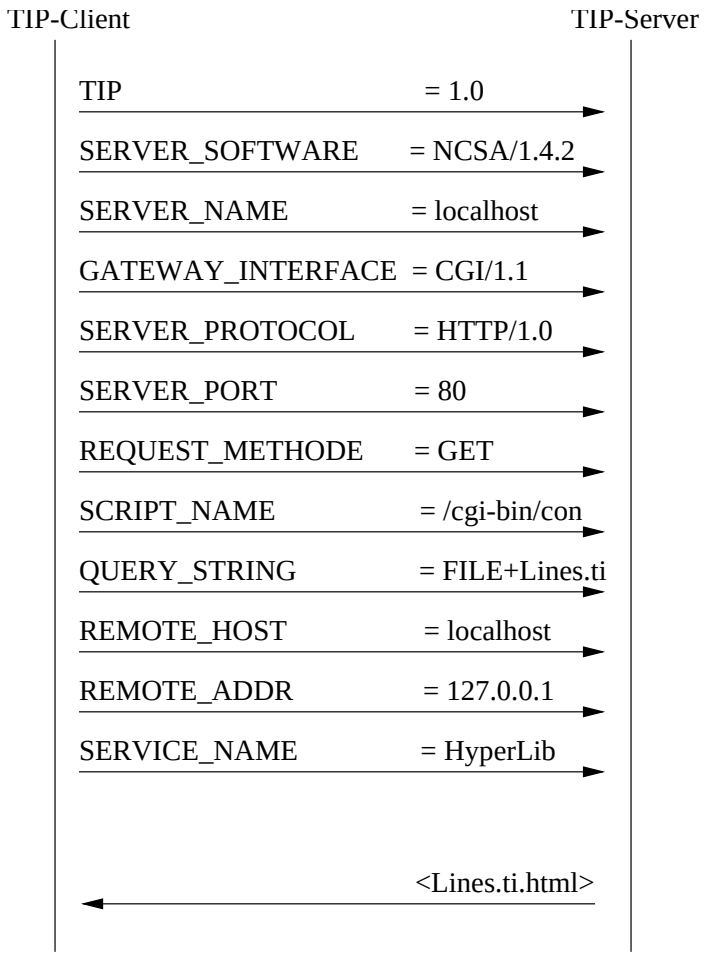

Abbildung 10: Beispielkommunikation zwischen TIP-Client und TIP-Server

Das Modul *TipServer* implementiert einen generischen TIP-Server. Es können beliebig viele TIP-Serverobjekte erzeugt werden. Jedes TIP-Serverobjekt belegt einen Netzwerkport. Bei einem Serverobjekt können beliebig viele Servicefunktionen angemeldet werden.

Die vom TIP-Client an den TIP-Server übergebene Menge von benannten Zeichenketten werden der Servicefunktion als Parameter in Form einer Tabelle übergeben. Es ist Aufgabe der Servicefunktion, die für sie wichtigen Parameter aus der Tabelle auszuwählen. Eine eindeutige Zuordnung von Namen zu Werten ist möglich, da das Modul *TipServer* mit Hilfe von HashTabellen implementiert wurde. Der Schlüssel eines jeden Eintrags einer Tabelle ergibt sich aus dem Namen des Eintrags. Ist ein Name nicht eindeutig, so ist auch sein Schlüssel nicht eindeutig, was zur Folge hat, das die Funktion zum Hinzufügen eines Eintrags zu einer Hashtabelle eine Ausnahme generiert.

```
interface TipServer
(* interface for server of the Tycoon Interface Protocol (TIP) *)
import hashTable
export
  Let E = Tuple name :String  value :String end

  T <: Ok
  (* type of server instance *)

  create(port :Int) :T
  (* instantiate a server object and allocate a network 'port' *)

  delete(:T) :Ok
  (* free a server object and deallocate network port *)

  activate(:T) :Ok
  (* activate server object and wait for connections *)

  addService(:T  name :String  serv(valTable :hashTable.T(E String)):String) :Ok
  (* add a servicefunction to a serverobject *)

end;
```

Abbildung 11: TIP-Serverschnittstelle

Abbildung 12 zeigt ein Beispiel für die Erzeugung und Verwendung eines TIP-Servers. Dieses Beispiel ist für die Verwendung in einem *Tycoon*-TopLevel gedacht. Zu Anfang wird ein Serverobjekt erzeugt (*ts*), dieses belegt nach seiner Erzeugung einen Netzwerkport. Dem Serverobjekt wird daraufhin eine Servicefunktion (*webService*) hinzugefügt. Nach der anschließenden Aktivierung wartet das Serverobjekt auf Verbindungsanfragen von Clients. Sollte eine Ausnahme bei der Bearbeitung einer Servicefunktion auftreten, so kehrt die Serveraktivierungsfunktion zurück und das nächste Kommando wird bearbeitet. In diesem Fall wird das Serverobjekt gelöscht.

Der hier implementierte Server für TIP ist nicht in der Lage, mehrere Verbindungsanfragen von Clients gleichzeitig zu bearbeiten. Er stellt auch keine Funktionen zur Verfügung, um zu ermitteln, welche Servicefunktionen schon vorhanden sind.

```
(* Definition einer Servicefunktion *)
let webService(valTable :hashTable.T(tipServer.E String)) :String =
    begin
    .
    .
    .

    end;

(* Erzeugen eines Serverobjektes mit Port = 6001 *)
let ts = tipServer.create(6001);

(* Hinzufügen einer Servicefunktion *)
tipServer.addService(ts "webServ" webService);

(* Aktivieren des Serverobjektes *)
tipServer.activate(ts);

(* Löschen des Serverobjektes und Freigeben des Ports *)
tipServer.delete(ts);
```

Abbildung 12: Beispiel für einen TIP-Server

4.3 Der Client

Zur Kommunikation mit dem TIP-Server wird ein in der Programmiersprache *C* implementierter Client verwendet. Tabelle 2 zeigt die Menge der möglichen Optionen des Clients.

Option	Beschreibung
-help	Ausgabe einer Parameterbeschreibung
-debug	Ausgabe von Debuginformation
-port	Spezifikation des Server Ports
-server	Spezifikation des Servers
Name / Wert	Benannte Parameter für den Service

Tabelle 2: Parameter des TIP-Clients

Ein Beispielaufruf sieht folgendermaßen aus:

```
tipClient -port 6001 -server dbis11 SERVICE_NAME MeinService Wert1 1 Wert2 2
```

Durch die Optionen *port* und *server* wird der Server ausgewählt. Die angegebene Menge von benannten Werten wird an den Server übertragen. Anhand des Wertes mit Namen *SERVICE_NAME* wählt der Server die Servicefunktion aus. Das Ergebnis des Aufrufs der Servicefunktion wird an den TIP-Client und von diesem an seinen Aufrufer zurückgegeben.

5 Erzeugung von Hypertext

Die Erzeugung von Hypertext für eine Visualisierungstechnologie gliedert sich in zwei Teile. Die Bezugsinformationen, die zusätzlich zu den *TL*-Programmbeschreibungen in expliziter Form benötigt werden, müssen aus den Programmbeschreibungen extrahiert werden. Die Programmbeschreibungen sollen so verändert werden, daß sie als Verweise markierte Textstellen beinhalten. Diese Verweise sollen Textstellen anderer Programmbeschreibungen, gemäß der Bezugsinformationen, referenzieren.

Die Trennung zwischen der Erzeugung von Hypertext und der Extrahierung von Bezugsinformationen erlaubt es, Hypertext für verschiedene Visualisierungstechnologien zu generieren.

Der in diesem Zusammenhang veränderte Übersetzer erzeugt während des Übersetzens zu jeder *TL*-Programmbeschreibung eine Menge von Bezugsinformationen. Diese Vorgehensweise scheint auch im Hinblick auf zukünftige Ergänzungen des *Tycoon*-Systems bezüglich der Programmbeschreibungs- und Objektdateiverwaltung sinnvoll. Programmbeschreibungen und Programmobjekte sollen dann im *Tycoon*-Objektspeicher abgelegt werden.

Themen dieses Kapitels sind die Erzeugung von Bezugsinformationen, die Verwaltung von Bezugsinformationen in Dateien und die Erzeugung von Hypertext aus *TL*-Programmbeschreibungen und Bezugsinformationen.

5.1 Die Verwaltung von Bezugsinformationen

Die Bezugsinformationen, die der Übersetzer während des Übersetzens sammelt, werden vom Modul *RefDB* als Textpositionsrelationen verwaltet (vgl. Abschnitt 1.3). Diese Informationen beschreiben jene Textpositionen der Programmbeschreibungen (in der Regel sind dies Bezeichner), die andere Textpositionen referenzieren. Für jede übersetzte Datei mit Programmbeschreibungen wird eine Datei mit Textpositionsrelationen erzeugt. Beziehungen, die Textstellen unterschiedlicher Programmbeschreibungsdateien verbinden, werden in den zugehörigen Textpositionsrelationendateien gespeichert.

Die Schnittstelle des Verwaltungsmoduls für Textpositionsrelationen gliedert sich in zwei Kategorien:

- Funktionen zur Verwaltung von in Dateien gespeicherten Textpositionsrelationen
- Funktionen zur Ermittlung von Informationen über Textpositionsrelationen

5.1.1 Verwaltung von Dateien mit Textpositionsrelationen

Textpositionsrelationen können in Dateien gespeichert werden. Die Namen der vom Übersetzer erzeugten Dateien mit Textpositionsrelationen haben einen Bezug zu den Namen der *TL*-Programmbeschreibungen, zu denen diese Relationen generiert wurden. Sie werden genauso benannt wie diese, erweitert um eine Extension ('.ddb').

Das Modul *RefDB* stellt Funktionen zur Verfügung, um Textpositionsrelationen in Dateien zu speichern und aus Dateien zu lesen. Textpositionsrelationen können auch zurückgesetzt werden.

Da die Bezugsinformationen verteilt gespeichert werden sollen, d.h. eine Textpositionsrelation für jede Programmbeschreibung (vgl. Abschnitt 2.1.3), kann es auch Beziehungen zwischen Textpositionsrelationen geben. Um diese Beziehungen zu verwalten, wird eine Funktion benötigt, die alle jene Tupel einer Relation löscht, die eine andere Relation referenzieren. Eine weitere Funktion ermittelt alle abhängigen Relationen einer Relation. Mit Hilfe dieser beiden Funktionen ist es möglich, alle Referenzen auf eine Textpositionsrelation zu löschen.

5.1.2 Anfragen über Textpositionsrelationen

Es ist möglich einfache Anfragen über Textpositionsrelationen zu stellen. Für die Darstellung einer Programmbeschreibung als Hypertext werden alle jene Textpositionen der Programmbeschreibung benötigt, die in Beziehung mit Textpositionen dieser und anderer Programmbeschreibung stehen. Hierfür gibt es eine Funktion, die alle referenzierten und alle referenzierenden Textpositionen einer Programmbeschreibung ermittelt. Der Name der Funktion ist *getPos*.

Um eine Hypertextseite mit allen mit einer Textposition in Bezug stehenden Textpositionen zu erstellen (vgl. Abschnitt 5.4), werden alle diese Textposition referenzierden Textpositionen benötigt. Die Funktion *queryPos* ermittelt alle diese Textpositionen.

5.1.3 Aufbau der Verwaltungsdateien

Um Textpositionsrelationen persistent zu machen, werden Dateien verwendet. Eine Textpositionrelation ist eine binäre Relation über einer Menge von Textpositionen. Jedes Tupel einer Textpositionsrelation beinhaltet also zwei Textpositionen. Eine Textposition einer Programmbeschreibungsdatei wird beschrieben durch den Dateipfad sowie die Position innerhalb der Datei. Die Position wird durch die Zeile und die Spalte ausgedrückt.

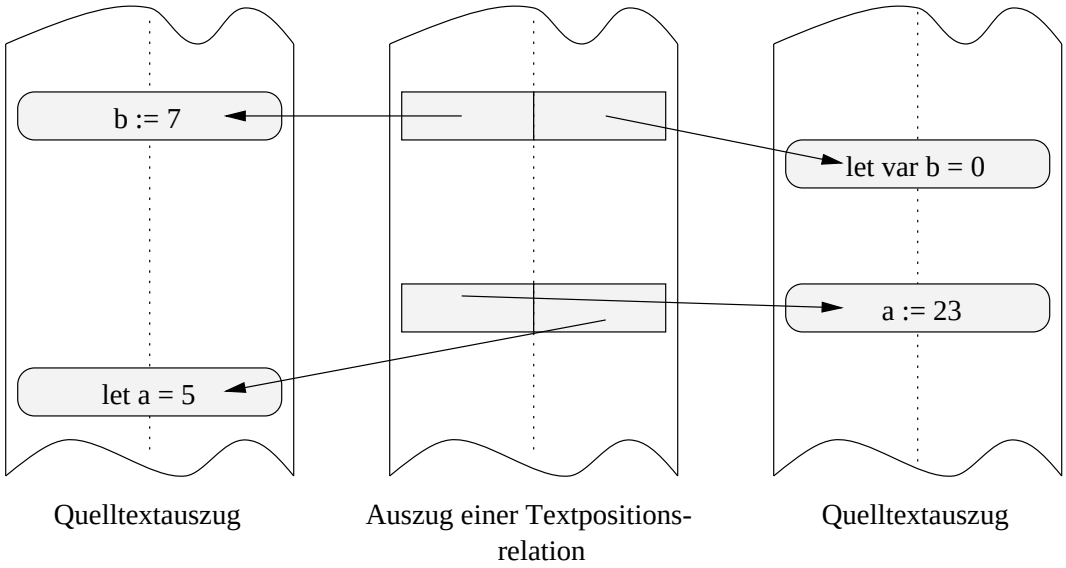

Abbildung 13: Auszüge eines Quelltextes und der zugehörigen Textpositionsrelation

Ein Tupel einer Textpositionsrelation hat folgendes Aussehen:

```
[Dateipfad]\n[Zeile]\n[Spalte]\n
[Dateipfad]\n[Zeile]\n[Spalte]\n
```

Es folgt ein Beispiel für ein Tupel einer Textpositionsrelation:

```
/local/nse/teamware/tycoon/tycoon/tycoon/src/stdenv/Iter.ti \n 1 \n 11 \n
/local/tw1/ramme/tl/tycoon/src/refLib/RefDB.ti \n 7 \n 24 \n
```

Im Beispiel wird das Wort in der ersten Zeile, in der elften Spalte der Schnittstellenbeschreibung *Iter.ti* durch das Wort in der siebten Zeile, der vierundzwanzigsten Spalte der Schnittstellenbeschreibung *RefDB.ti* referenziert. Diese beiden Worte stehen in Beziehung miteinander.

5.2 Modifikation des Übersetzers

Da der *TL*-Übersetzer die Textpositionsrelationen während des Übersetzens erzeugen soll (vgl. Abschnitt 2.1), mußten an ihm einige Änderungen vorgenommen werden.

Der Übersetzer ermittelt referentielle Beziehungen zwischen Objekten der zu übersetzenden Programmbeschreibungen. Diese referentiellen Beziehungen wer-

den innerhalb der Programmbeschreibungen mit Hilfe von Bezeichern ausgedrückt. Um einem im Eingabestrom erkannten Bezeichner einem anderen Bezeichner zuzuordnen, wird in einer Symboltabelle nach einem Bezeichner mit gleichem Aussehen gesucht. Wird so ein Bezeichner gefunden, so ist ein Bezug hergestellt. Bezeichner werden in die Symboltabelle bei einer Deklaration bzw. Definition eingetragen.

Der Übersetzer wurde so geändert, daß er die erfolgreiche Zuordnung zweier Bezeichner zueinander in eine Textpositionsrelation als Bezug einträgt. [3] Diese Textpositionsrelation beinhaltet also alle erfolgreich ermittelten Beziehungen zwischen Bezeichern.

Beschreibungen von Schnittstellen, Moduln und Bibliotheken werden in separaten Dateien verwaltet. Der Übersetzer wurde dahingehend geändert, daß er bei der Öffnung einer zu übersetzenden Datei die Textpositionsrelation, in der er die Beziehungen der Bezeichner sammelt, zurücksetzt.

Das Ergebnis der Modifikationen am Übersetzer ist, daß der Übersetzer nun zu jeder Schnittstellen-, Modul- oder Bibliotheksbeschreibung eine Textpositionsrelation erzeugt.

5.3 Modifikation von Programmbeschreibungen

Die in der Programmiersprache *TL* formulierten Programmbeschreibungen des *Tycoon*-Systems sollen in Hypertext konvertiert werden. Um dies zu erreichen, müssen Teile der Programmbeschreibungen markiert werden. Eine Textstelle wird markiert, indem sie eingeklammert wird. Die Art der Klammerung bestimmt die Art der Markierung.

Da Programmbeschreibungen durch die Konvertierung ihre Formatierung nicht verlieren sollen, ist es nötig, die Menge der Textstellen, die Formatierungsanweisungen beinhalten, von der Menge der Textstellen, die für die Konvertierung relevant sind, zu trennen. Diese Trennung erlaubt es, Textstellen einer Programmbeschreibung ohne Veränderung der Formatierung der Programmbeschreibung zu modifizieren.

Das Modul *Lines* erlaubt die Modifikation von Texten ohne Veränderung der Formatierung.

[3] Eine Programmbeschreibung kann beliebig viele gleiche Bezeichner haben, zwei gleiche Bezeichner sind jedoch nie identisch

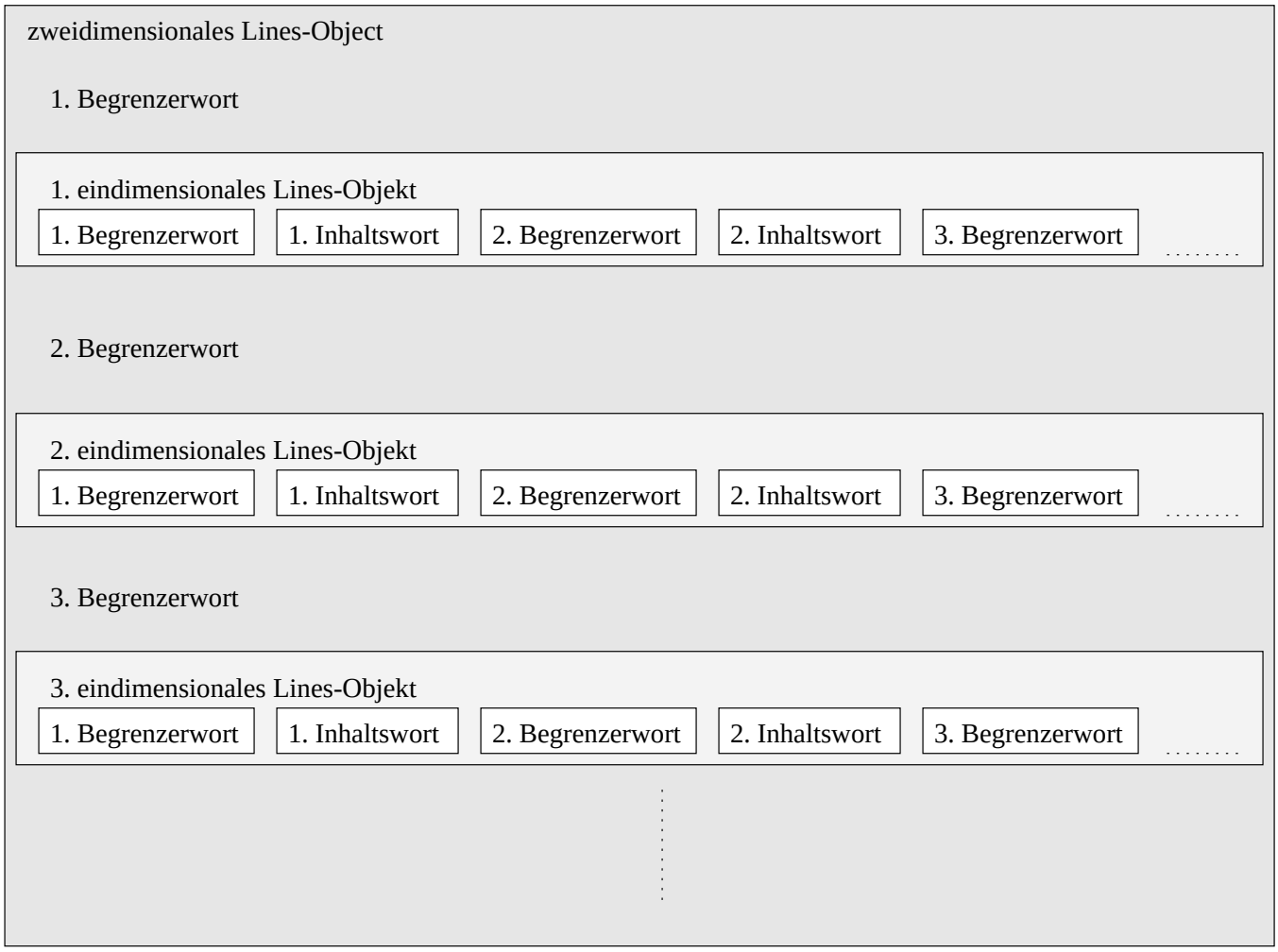

Abbildung 14: Ein Lines-Objekt

5.3.1 Lines-Objekte und Zeichenketten

Texte sind Folgen von Zeichenketten. Formatierungsanweisungen innerhalb von Zeichenketten werden durch Formatierungszeichen ausgedrückt. Um die Formatierung eines Textes vom Inhalt zu trennen, wird der Text in eine Folge von Zeichenketten zerlegt. Jedes Element der Folge beschreibt entweder einen Teil der Formatierung oder einen Teil des Inhaltes des Textes. Die Folge ist eine alternierende Folge von Formatierungszeichenketten und Inhaltszeichenketten. Begrenzer, die Worte innerhalb von Zeichenketten trennen, sind Formatierungszeichen.

Da innerhalb des *TL*-Übersetzers zwischen zwei Arten von Formatierungsanweisungen unterschieden wird (Leerzeichen, Zeilenrücklauf), können bei der Konvertierung einer Zeichenkette in ein Lines-Objekt wahlweise eine oder zwei Mengen mit Formatierungszeichen angegeben werden. Werden zwei Mengen von Forma-

tierungszeichen angegeben, so ist die resultierende Folge eine alternierende Folge von Zeichenketten aus Formatierungszeichen der einen Formatierungszeichenmenge und alternierenden Folgen von Zeichenketten der anderen Formatierungszeichmenge und Zeichenketten der Inhaltszeichenmenge. Abbildung 14 zeigt ein solches Lines-Objekt.

Lines-Objekte können wieder in Zeichenketten konvertiert werden. Werden keine Formatierungszeichen verändert, so hat die resultierende Zeichenkette die gleiche Formatierung wie die ursprüngliche.

5.3.2 Setzen von Worten

Ein Lines-Objekt ist eine alternierende Folge von Zeichenketten aus Formatierungszeichen und Nichtformatierungszeichen. Diese Zeichenketten bilden Worte. Lines-Objekte erlauben es, Worte abzufragen und zu setzen. Im Falle von zwei Mengen von Formatierungszeichen sind die Nichtformatierungsworte des Lines-Objektes wiederum Lines-Objekte.

5.3.3 Prä- und Postfixe

Lines-Objekte erlauben es, für Worte aus Nichtformatierungszeichen Prä- und Postfixe zu setzen. Bei einer Konvertierung eines Lines-Objektes in eine Zeichenkette werden diese vor bzw. hinter dem entsprechenden Wort eingefügt. Dies erleichtert das Markieren von Textstellen.

5.4 Die Erzeugung von HTML-Seiten

Die Erzeugung von HTML-Seiten wird im Modul *QueryHTML* implementiert.

Jede Textposition einer Progammbeschreibung kann mit einer Menge von anderen Textpositionen in Beziehung stehen. Eine Textpositionsrelation ist also eine **n** zu **m** Relation. Eine als Verweis markierte Textposition einer nach HTML konvertierten Programmbeschreibung kann also auf **n** andere Textpositionen verweisen. Um dies visualisieren zu können, wird eine Verweisseite eingeführt, welche alle mit einer Textpositionsrelation in Beziehung stehenden Textpositionen auflistet.

Die im Rahmen dieser Studienarbeit vom *TL*-Übersetzer erzeugten Textpositionsrelationen sind nur 1 zu n Relationen, da nur Beziehungen zwischen Bezeichnern beachtet werden. Würden auch beziehungen zwischen Typen analysiert werden, so ergäben sich n zu m Relationen. Ein Typ kann mit beliebig vielen anderen Typen in einer Subtypbeziehung stehen.

Es gibt also folgende Arten von HTML-Seiten:

- Seiten mit einer Liste aller Zeilen von nach HTML konvertierten Programmbeschreibungen, die Verweise auf eine Quelltextposition beinhalten (Verweisseiten; siehe Abb. 15).
- Nach HTML konvertierte Programmbeschreibungen. Die in Beziehung stehenden Textposition werden als Verweise auf Verweisseiten markiert (siehe Abbildung 16).

5.4.1 Verweisseiten

Eine Funktion des Moduls *QueryHTML* erstellt HTML-Seiten mit Auszügen aus Programmbeschreibungen, in denen eine angegebene Textposition referenziert wird. Sortiert werden diese Auszüge nach den Namen der zugehörigen Programmbeschreibungen und der jeweiligen Entfernung vom Anfang der Programmbeschreibung.

Als Hilfe für die Erzeugung dieser Seiten mit referenzierenden Textstellen verwendet diese Funktion eine HTML-Rahmenseite (vgl. Anhang D), in welcher jene Positionen markiert sind, die durch Programmbeschreibungsauszüge ersetzt werden sollen. Die Markierungen werden durch spezielle Schlüsselworte kenntlich gemacht. Tabelle 3 zeigt die möglichen Schlüsselworte.

Schlüsselwort	Beschreibung
#NAME#	Wort an der referenzierenden Textposition
#DEFINITION#	Zeile der referenzierten Textposition
#USAGE#	Weitere Zeilen mit Referenzen auf dieselbe Textposition

Tabelle 3: Schlüsselworte der Rahmenseite

Zeilen aus Programmbeschreibungen, die in der Verweisseite zu sehen sind, werden vor ihrer Darstellung auch nach HTML konvertiert. Die Verweise innerhalb dieser Zeilen verhalten sich wie die Verweise innerhalb der zugehörigen konvertierten Programmbeschreibung. Um die zugehörige Programmbeschreibung direkt anspringen zu können, wird jede solche Zeile mit einem Verweis auf die entsprechende Zeile der zur Programmbeschreibung zugehörigen HTML-Seite versehen.

Abbildung 15: Beispiel einer Verweisseite

5.4.2 Quelltext als HTML

Eine andere Funktion des Moduls *QueryHTML* konvertiert eine Programmbeschreibung nach HTML. Diese Funktion ermittelt zuerst alle jene Textpositionen die in Beziehung mit anderen Textpositionen stehen. Die Worte an diesen Textpositionen werden nun durch Verweise auf Verweisseiten ersetzt. Die Verweisseiten listen alle mit dieser Textposition in Beziehung stehenden Textposition auf (vgl. Abb. 16).

5.5 Das HTTP-Server Gateway

Damit ein WWW-Client auf die in *TL* implementierte Funktionalität zur Erzeugung von HTML-Seiten zugreifen kann, wird ein CGI-Gateway benötigt. Dieses Gateway stellt die Verbindung zwischen dem HTTP-Server und dem *Tycoon*-Server her. Das Gateway liest die Umgebungsvariablen aus und übergibt diese zusammen mit einer Spezifikation des Servers und des Netzwerk-Ports an den TIP-Client (vgl. Kapitel 4.3). Die Adresse des Servers und die Nummer des Ports werden innerhalb des Gateways festgelegt (siehe Anhang C und Abb. 17).

5.6 Verwendung des Werkzeugs

Um das während dieser Studienarbeit implementierte System benutzen zu können, fehlt noch eine Verbindungsfunktion, die die Funktionen des Moduls zur Erzeugung von HTML (*queryHTML*) mit dem *TIP*-Server verbindet. Diese Funktion (*webService*) ist Teil des *TL*-Skriptes (*setUp.tyc*), welches den *TIP*-Server instantiiert. Die Funktion erkennt anhand der übermittelten Parameter des *TIP*-Clients den Typ von HTML-Seite, der zu erzeugen ist.

Das Gateway, das HTTP-Server und TIP-Server verbindet, gehört in das Verzeichnis der ausführbaren Dateien des HTTP-Servers.

Es folgen die Beschreibung für den Aufbau einer URL für den Zugriff auf einen TIP-Server und ein Beispiel:

`http://dbis9/`*`gateway`*`/FILE+`*`Programmbeschreiungsdatei`*

`http://dbis9/tipConnect/FILE+Lines.ti`

Abbildung 17 zeigt HTTP-Client und *Tycoon*-Server.

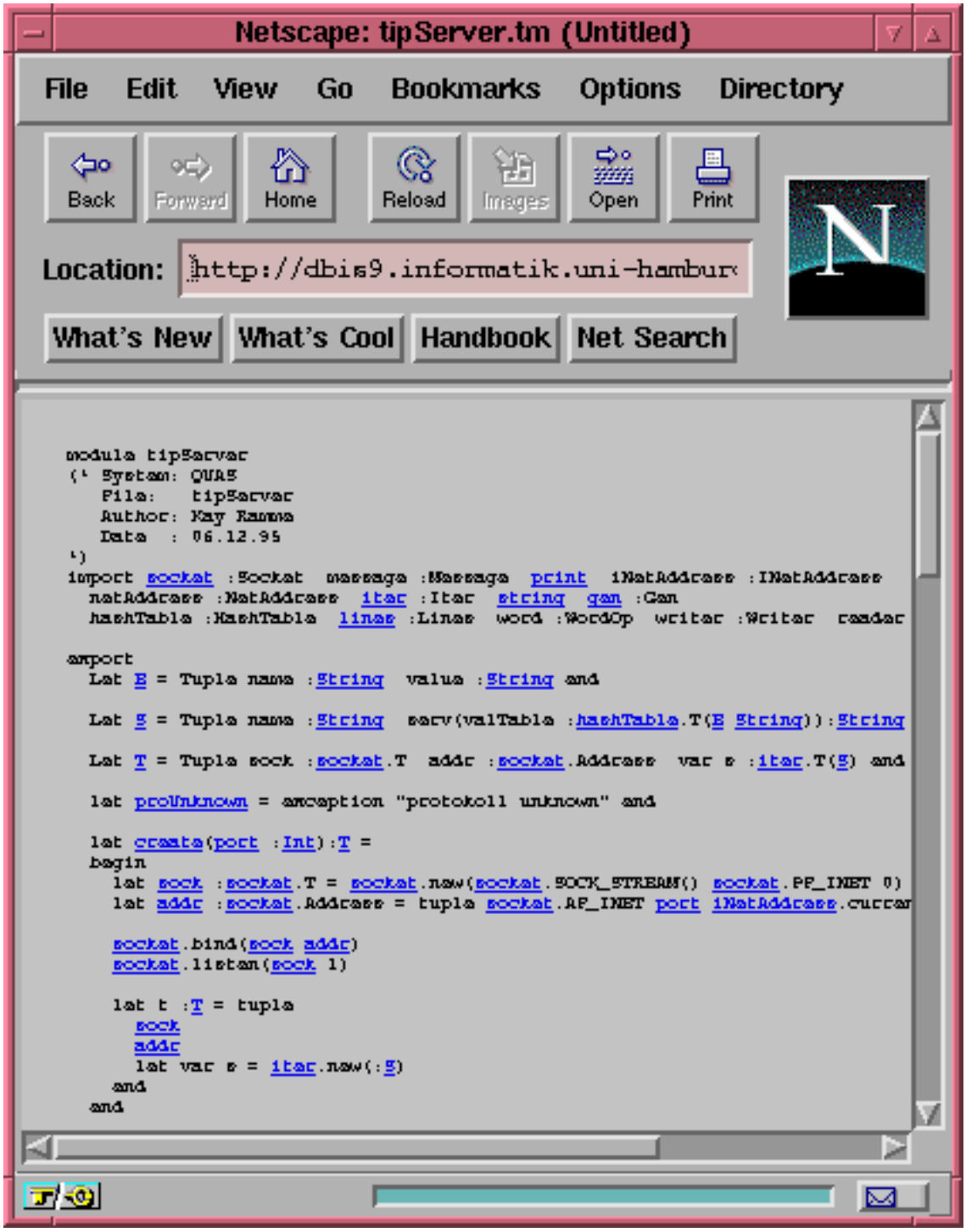

Abbildung 16: Beispiel einer Quelltextseite

6 Zusammenfassung und Ausblick

6.1 Ansätze für Verbesserungen

Folgende Punkte sind mögliche Ansätze für Verbesserungen:

- Paralelle Bearbeitung der Anfragen mehrerer Clients. Der TIP-Server sollte dahingehend erweitert werden, daß er mehrere Anfragen gleichzeitig bearbeiten kann. Dies wäre beispielsweise mit der Erzeugung eines Threads für jede Anfrage möglich.
- Typisierung von Textpositionen und Ergänzung der Arten von verwendeten Bezugsinformationen. Eine Textposition kann dann unterschiedliche Beziehungen zu anderen Textpositionen haben. Der Bezeichner eines Typs einer in *TL* verfaßten Programmbeschreibung kann z.B. einen Verweis auf eine Verweisseite mit allen diesen Typ referenzierenden Textpositionen haben. Durch obige Ergänzung könnte er auch noch auf eine Verweisseite mit einer Liste aller Subtypen verweisen.
- Verwendung von Textstellenrelationen statt Textpositionsrelationen. Dies ist abhängig von der Art der Symbolbeschreibungen der Symboltabellen des Übersetzers. Zur Zeit werden nur Zeile und Spalte des ersten Zeichens eines Symbols gespeichert. Es fehlt die Länge des Symbols. Bei Verwendung von Textstellenrelationen könnten Programmbeschreibungen leichter konvertiert werden, d.h. es wäre keine Zerlegung in Wortfolgen mit Hilfe von Begrenzern mehr nötig.

6.2 Übersicht über die Implementation

Das im Rahmen dieser Studienarbeit entwickelte Projekt gliedert sich in mehrere Teile, die in *TL*, *C* und als Shellskript implementiert sind.

- Das in *TL* implementierte Modul *RefDB* verwaltet die Bezugsinformationen als Textpositionsrelationen.
- Ein modifizierter *Tycoon*-Übersetzer erzeugt während des Übersetzens von Modulen, Schnittstellen und Biliotheken Bezugsinformationen.
- Ein in *TL* implementiertes Modul (*Lines*) ermöglicht es, Programmbeschreibungen in veränderbare Einheiten zu zerlegen. Der Programmtext kann nach Manipulation der Einheiten wieder in seiner ursprünglichen Formatierung hergestellt werden.

- Ein in *TL* implementiertes Modul (*QueryHTML*) erzeugt aus Bezugsinformationen und *TL*-Programmbeschreibungen Hypertext im Format von HTML.
- TIP-Server und TIP-Client ermöglichen es, Funktionen, die sich innerhalb eines *Tycoon*-Objektspeichers befinden, von Programmen aus aufzurufen, die sich nicht innerhalb des Objektspeichers befinden. Diese Funktionalität wird benötigt, um den HTTP-Server mit den in *TL* programmierten Funktionen zur Erzeugung von Hypertext zu verbinden.
- Ein *Tycoon*-Skript instantiiert einen *Tycoon*-Server mit einem Service zur Erzeugung von HTML.
- Ein Shellskript dient als Gateway zwischen HTTP-Server und TIP-Client.

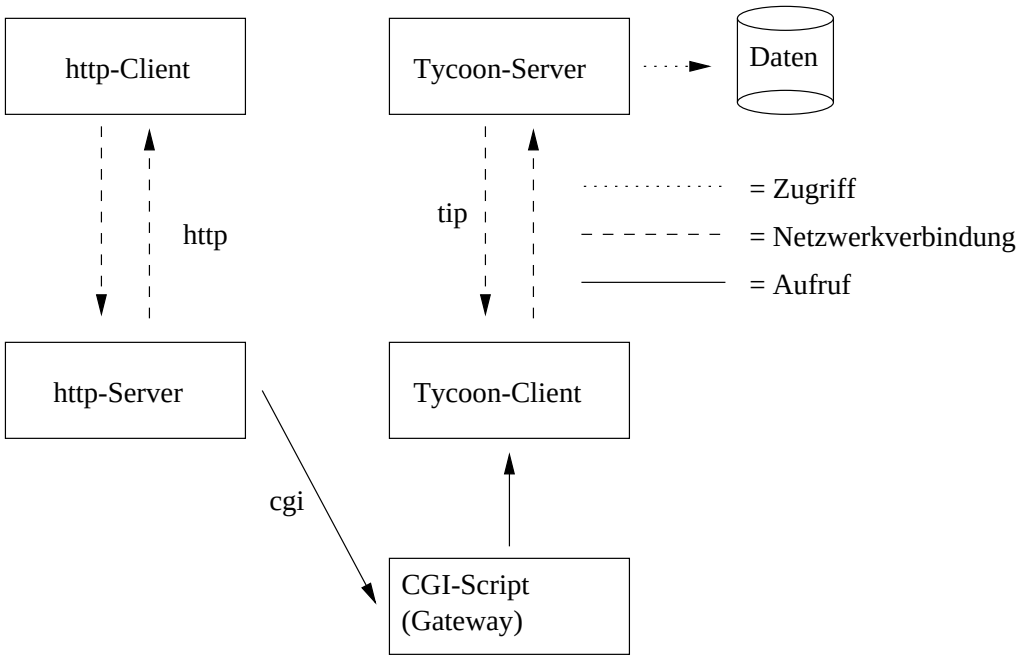

Abbildung 17: Kommunikation zwischen HTTP-Client und *Tycoon*-Server

Abbildung 17 zeigt die Kommunikation zwischen HTTP-Client und *Tycoon*-Server. Der HTTP-Client baut zunächst eine Verbindung zu einem HTTP-Server auf. Der Server startet ein Shellskript als Gateway, wobei vom HTTP-Client an den Server übergebene Parameter an das Gateway durchgereicht werden. Das Gateway ruft den TIP-Client auf und übergibt als zusätzliche Parameter den Namen des anzusprechenden TIP-Servers sowie den Namen der vom TIP-Server aufzurufenden Servicefunktion. Der TIP-Client baut eine Verbindung zu einem TIP-Server auf und transferiert seine Parameter. Das Ergebnis des Aufrufs der

Servicefunktion wird vom TIP-Server über den TIP-Client und über den HTTP-Server an den HTTP-Client zurückgegeben.

7 Zur Entwicklung

Es hat bei der Entwicklung dieses Projektes relativ lange gedauert, um zu erkennen, daß es sich bei den zusätzlich zu den Programmbeschreibungen benötigten Informationen um Relationen über Textpositionen handelt, da die Semantik dieser Zuordnungen im Übersetzer spezifiziert ist.

Der erste Versuch mit Scopes und Scopepfaden zu arbeiten hatte nur begrenzten Erfolg. Ein Bezeichner einer Programmbschreibung wird dabei mit Hilfe eines Pfades, vergleichbar mit einem Pfad eines hierarchischen Dateisystem, beschrieben. Dieser Pfad setzt sich aus den Namen der Scopes, die zu diesem Bezeichner führen, zusammen. Probleme bereiten anonyme Scopes, da zwei gleiche Bezeichner zweier anonymer Scopes, die sich im gleichen übergeordneten Scope befinden, nicht mehr untschieden werden können. Es ist auch nicht gelungen die Scopepfade innerhalb des Übersetzers wiederzufinden, dies machte es unmöglich ihn dahingehend zu verändern, daß er Bezugsinformationen erzeugt.

A Tycoon Interfaces

A.1 TipServer.ti

```
interface TipServer
(* interface for server of the Tycoon Interface Protocol (TIP) *)
import hashTable
export
  Let E = Tuple name :String  value :String end

  T <: Ok
  (* type of server instance *)

  create(port :Int) :T
  (* instantiate a server object and allocate a network 'port' *)

  delete(:T) :Ok
  (* free a server object and deallocate network port *)

  activate(:T) :Ok
  (* activate server object and wait for connections *)

  addService(:T  name :String  serv(valTable :hashTable.T(E String)):String) :Ok
  (* add a servicefunction to a serverobject *)

end;
```

A.2 RefDB.ti

```
interface RefDB
(* interface for a modul to manage a cross reference database *)
import :Source  iter
export
        ext :String

        Let RT = Tuple pos1 :Source.Position  pos2 :Source.Position end

        T <:Ok
        (* type *))

        create(path :String) :T
        (* create a database saved at path 'path' *)

        clear(:T) :Ok
        (* reset a database *)

        writeToFile(:T) :Ok
        (* save a database *)

        readFromFile(path :String) :T
        (* load a database from file at path 'path' *)

        clearPart(:T  path :String) :Ok
        (* delete all references to database at path 'path' *)

        dependents(:T) :iter.T(String)
        (* get list of depending databases *)

        addToCurrent(:T :RT) :Ok
        (* add a reference to a database)

        queryPos(pos :Source.Position) :iter.T(Source.Position)
        (* get a list of all positions referencing position 'pos' *)

        getPos(:T) :iter.T(RT)
        (* get a list of all referenced positions of a database *)

end;
```

A.3 Lines.ti

```
interface Lines
(* interface for a modul to separate lines and words of a string *)
import iter
export
        error :Exception end

        T <:Ok

        e :T

        create(str :String  dls :String  mFlg :Bool) :T
        (* convert string to lines object with word delimiters of 'dls'
           mFlg = true: a stream of delimiters counts once,
           mFlg = false: each delimiter in a delimiter stream counts *)

        toString(:T) :String
        (* convert lines object to string *)

        create2(str :String  lneDl :String  wrdDl :String) :T
        (* convert string to lines object with word delimiter of 'wrdDl'
           and lines delimiters of 'lneDl' *)

        words(:T) :Int
        (* count words *)

        get(:T  idx :Int) :String
        (* get word number 'idx' *)

        get2(:T  lneIdx :Int  wrdIdx :Int) :String
        (* get word number 'wrdIdx' of line 'lneIdx' *)

        set(:T  idx :Int  str :String) :Ok
        (* set word number 'idx' *)

        set2(:T  lneIdx :Int  wrdIdx :Int  str :String) :Ok
        (* set word number 'wrdIdx' of line 'lneIdx' *)

        getDL(:T  idx :Int) :String
        (* get delimiter word number 'idx' *)

        getDL2(:T  lneIdx :Int  wrdIdx :Int) :String
        (* get delimiter word number 'wrdIdx' of line 'lneIdx' *)
```

setDL(:T idx :Int str :String) :**Ok**
(set delimiter word number 'idx' *)*

setDL2(:T lneIdx :Int wrdIdx :Int str :String) :**Ok**
(set delimiter word number 'wrdIdx' of line 'lneIdx' *)*

setPreFix(:T idx :Int str :String) :**Ok**
(set prefix of word number 'idx' *)*

setPreFix2(:T lneIdx :Int wrdIdx :Int str :String) :**Ok**
(set prefix of word number 'wrdIdx' of line 'lneIdx' *)*

setPostFix(:T idx :Int str :String) :**Ok**
(set postfix of word number 'idx' *)*

setPostFix2(:T lneIdx :Int wrdIdx :Int str :String) :**Ok**
(set postfix of word number 'wrdIdx' of line 'lneIdx' *)*

getWrdIdxByColIdx(:T lneIdx :Int colIdx :Int) :*Int*
(convert column index 'colIdx' to word index at line 'lneIdx' *)*

end;

A.4 QueryHTML.ti

```
interface QueryHTML
(* interface for a modul to convert tl sources to html *)
import :Source
export
        queryFile(path :String  scriptName :String) :String
        (* convert program description at path 'path' to html
           'scriptName' is the name of the gateway to place in links *)

        queryPos(pos :Source.Position  scriptName :String) :String
        (* create reference overview page of posisition 'pos'
           'scriptName' is the name of the gateway to place in links *)

end;
```

A.5 Gen.ti

```
interface Gen
(* interface for a modul with string functions *)
export
        genExc :Exception end

        strLoad(path :String) :String
        (* load a string from file *)

        strSave(path :String  str :String) :Ok
        (* save a string to file *)

        strReplace(str :String  wrd :String  new :String) :String
        (* replace all occurrences of wrd with new *)

        strRplN(str :String  idx :Int  len :Int  is :String) :String
        (* replace range from idx to idx + len - 1 with is)

        strLines(str :String) :Int
        (* count lines *)

        strLoadLine(path :String  row :Int) :String
        (* load line row of file *)

        strLoadLines(path :String) :Array(String)
        (* load string from file and convert it to lines array *)

        eatSpc(str :String  dl :String) :String
        (* return str without leading whitespace *)

        getWord(str :String  offset :Int  dl :String) :String
        (* return word at offset `offset` *)

end;
```

B TipClient Parameter

Der Aufruf des Programms *TipClient* hat folgendes Format:

```
tipClient [-port <num>] [-server <a.b.c.d>] [-debug] [-help] [name value]*
```

Die Parameter zur Beschreibung des Servers und des Ports können auch als Wertepaare übergeben werden.

```
TIP_SERVER_PORT <num> TIP_SERVER_NAME <a.b.c.d>
```

C HTTP-Server Gateway zu *Tycoon*(Listing)

```
#!/bin/sh
#cgi-script
#18.04.1996 Kay Ramme

pairs=""

if test $SERVER_SOFTWARE
then
pairs="$pairs SERVER_SOFTWARE $SERVER_SOFTWARE"
fi

if test $GATEWAY_INTERFACE
then
pairs="$pairs GATEWAY_INTERFACE $GATEWAY_INTERFACE"
fi

if test $SERVER_PROTOCOL
then
pairs="$pairs SERVER_PROTOCOL $SERVER_PROTOCOL"
fi

if test $SERVER_PORT
then
pairs="$pairs SERVER_PORT $SERVER_PORT"
fi

if test $REQUEST_METHOD
then
pairs="$pairs REQUEST_METHOD $REQUEST_METHOD"
fi

if test $HTTP_ACCEPT
then
pairs="$pairs HTTP_ACCEPT $HTTP_ACCEPT"
fi

if test $PATH_INFO
then
pairs="$pairs PATH_INFO $PATH_INFO"
fi
```

```
if test $PATH_TRANSLATED
then
pairs="$pairs PATH_TRANSLATED $PATH_TRANSLATED"
fi

if test $SCRIPT_NAME
then
pairs="$pairs SCRIPT_NAME $SCRIPT_NAME"
fi

if test $QUERY_STRING
then
pairs="$pairs QUERY_STRING $QUERY_STRING"
fi

if test $REMOTE_HOST
then
pairs="$pairs REMOTE_HOST $REMOTE_HOST"
fi

if test $REMOTE_ADDR
then
pairs="$pairs REMOTE_ADDR $REMOTE_ADDR"
fi

if test $REMOTE_USER
then
pairs="$pairs REMOTE_USER $REMOTE_USER"
fi

if test $AUTH_TYPE
then
pairs="$pairs AUTH_TYPE $AUTH_TYPE"
fi

if test $CONTENT_TYPE
then
pairs="$pairs CONTENT_TYPE $CONTENT_TYPE"
fi

if test $CONTENT_LENGTH
then
```

```
pairs="$pairs CONTENT_LENGTH $CONTENT_LENGTH"
fi

pairs="$pairs $* SERVICE_NAME quasServ"

echo "Content-type: text/html"
echo ""

echo `/home/kay/projects/quas/tip/tipClient -debug -port 6001 \
      -server dbis11.informatik.uni-hamburg.de $pairs &`
```

D Rahmenseite für Verweisseiten

```
<HEAD>
<TITLE>Name: #NAME#</TITLE>
</HEAD>
<BODY>
<H1>Name: #NAME#</H1>
<P>
Definition:
<P>
#DEFINITION#
<P>
Verwendung:
<P>
<PRE>
#USAGE#
</PRE>
<P>
<HR>
<P>
Date:
<P>
</BODY>
```

E Literaturverzeichnis

Cricket Liu, Jerry Peek et al. Managing Internet Information Services. O'Reilly & Associates, Inc., 1994.

Reinhard Detering. UNIX Handbuch System V. Sybex 1989.

Florian Matthes, Sven Müßig. The Tycoon Language TL: An Introduction, DBIS Tycoon Report 112-93, Decmber 1993.

Gerald Schröder, Florian Matthes. Using the Tycoon Compiler Toolkit, DBIS Tycoon Report 061-92, Juni 1992.