# Entwicklungsmethodik für Kommunikationsprotokolle auf der Basis von Software Pattern

Prof.´ín Dr. Beate Commentz-Walter


FH Albstadt - Sigmaringen

## Schlüsselworte

Pattern *"General Distributed Proxy",* Communication Protocol Design, Prototyping, Software Development Process.

## Abstract

Ziel dieses Artikels ist eine Verbesserung des Entwicklungsprozesses von Kommunikationsprotokollen verteilter Systeme mit Entwurfsmustern.

Dazu wird hier das*"General Distributed Proxy"* Pattern beschrieben. Es ist ein Entwurfsmuster für die Kommunikation verteilter Objekte das das *"Distributed Proxy"* Pattern [SRG 02], [SRGA 02] im Sinne der Offenhaltung des Kommunikationsprotokolls verallgemeinert. Es dient somit als allgemeine Basis für den Entwurf eines Kommunikationsprotokolls und zur Verbesserung dieses Prozesses.

In diesem Artikel wird das *"General Distributed Proxy"* Pattern beschrieben, mit dem Distributed Proxy verglichen und seine Anwendung am Beispiel eines einfachen File Transfer Protokolls (FTP) demonstriert.

This paper aims to improve the process of communication protocol development using Software Pattern.

It presents a *"General Distributed Proxy"* pattern, a design pattern for distributed object communication based upon the *"Distributed Proxy"* Pattern [SRG 02], [SRGA 02]. Opposite to the *"Distributed Proxy"* pattern the *"General Distributed Proxy"* pattern does not predetermine the communication protocol of the distributed objects. The General Distributed Proxy serves as base of communication protocol design and as process improvement.

This paper describes the *"General Distributed Proxy"* pattern and compares it to *"Distributed Proxy"* pattern. In addition, an example protocol is presented. It is a primitive file transfer protocol (FTP).

## 1 Einleitung

### 1.1 Motivation

Nach der Normenreihe ISO 9000 ff zum Qualitätsmanagement ist die Verbesserung des Produktionsprozesses eines der wichtigsten Mittel zur Verbesserung der Produktqualität.

Insbesondere die Qualität von Software Produkten hängt in entscheidendem Maße vom Prozess ab, in dem diese Produkte entstanden sind. Darum leistet die objektorientierte Methodik mit der zugehörigen Beschreibungstechnik *UML* [OMG UML 1.3], geeigneten Entwurfsmustern und Schichtarchitektur einen wichtigen Beitrag zur Entwicklung von zuverlässigen und wartungsfreundlichen Systemen, vgl. auch [KRÄ 99] S. 1.

Software Produkte sind heute in zunehmenden Maße jedoch verteilte Systeme. Bei der Entwicklung von verteilten Systemen ist der Entwurf des Kommunikationsprotokolls ein wichtiger und besonders fehleranfälliger Teilprozess in einer komplexen Umgebung.

Darüber hinaus ist ein Anwendungstest oft erst nach Implementierung und Test des Protokolls möglich.

Der Entwurf eines Kommunikationsprotokolls wird durch die bekannten Entwurfsmuster *"Distributed Proxy"* Pattern [SRG 02], [SRGA 02] und *„Acceptor - Connector"* [BRS 00] S. 285 ff, *Communicator"* sowie *„Forwarder - Receiver"* [BMR 96] S 308 ff. grundsätzlich gut unterstützt.

Die Pattern gliedern ein Kommunikationsprotokoll in Schichten: die Anwendungs-, die Logische und die Physikalische Schicht, vgl. Abb. 1:

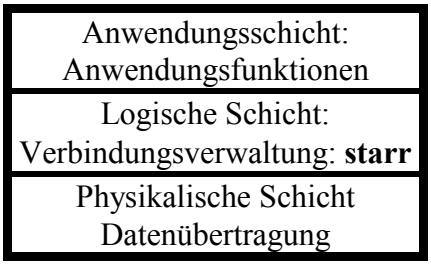


Abbildung 1: **Schichten eines Kommunikationsprotokolls**

So werden die Protokoll-Entscheidungen entsprechend schichtweise getroffen und nur die Implementierung der Anwendungsschicht wird für einen Anwendungstest benötigt.

Darüber hinaus liefern die Pattern den Entwurf für die Protokoll-Realisierung und das Protokoll selbst kann so durch die Spezifikation der Pattern-Klassen (Syntax und Semantik) dokumentiert werden.

Die Muster *„Acceptor - Connector"* und *„Forwarder - Receiver"* gliedern die Anwendung in nur 2 Schichten. Sie unterstützen den Entwurf von Kommunikationsprotokollen besonders gut, da sie die Entwurfsfreiheit für das Protokoll nicht einschränken. Allerdings muss die funktionale Klasse *„ServiceHandler"* der Anwendung nach

[BRS 00] S. 285 ff prinzipielle Kenntnisse über die von ihr benutzten Verbindungen zu anderen Objekten haben.

Das Distributed Proxy Pattern hingegen definiert zusätzlich die Klassen *„ClientSideProxy“* und *„ServerSideProxy“* und gliedert die Anwendung dadurch in 3 Schichten. Es bietet ein Maximum an Unabhängigkeit und Austauschbarkeit der Schichten sowie Transparenz des benutzten IPCs (Inter Prozess Kommunikationsmechanismus) für die oberste, funktionale Schicht.

Der Overhead einer zusätzlichen Schicht ist unbedeutend, da ein optimierender Compiler diesen wieder eliminiert.

Die funktionale Schicht kann als lokale Anwendung implementiert werden und erst danach zu einer verteilten Anwendung erweitert werden. Dadurch lässt sich das Pattern leicht in einen inkrementellen Entwicklungsprozess integrieren. Insbesondere der Anwendungstest kann zu einem frühen Zeitpunk durchgeführt werden.

Leider hat das Distributed Proxy Pattern einen Nachteil. In diesem Muster wird für jeden Methodenaufruf (jede Nachricht) eine Verbindung auf- und wieder abgebaut. Dadurch wird das Kommunikationsprotokoll der verteilten Objekte ungünstig vorausbestimmt. Die Anwendung wird zu häufigem Verbindungsauf- und -abbau genötigt. Ein funktional an die Anwendung angepasstes Kommunikationsprotokoll sollte dieses jedoch aus Performance Gründen vermeiden.

Wenn man beispielsweise einer Person mehrere Fragen stellen will, wird man auch nicht jedes mal zwischen zwei Fragen das Zimmer verlassen um es sofort anschließend wieder zu betreten.

Nur ein funktional angepasstes Konzept für Verbindungsauf- und -abbau ermöglicht eine optimale Netz-Nutzung. Damit ist die für den Verbindungsaufbau zuständige Schicht jedoch leider nicht mehr vollkommen unabhängig von den Funktionen der Anwendung.

Bei der FTP Anwendung des Internets [RFC 959] beispielsweise wird eine „Steuerverbindung“ für den Austausch der Kommandos verwendet. Diese Verbindung bleibt während der gesamten Sitzung erhalten.

Für jeden Datei Transfer wird zusätzlich eine eigene „Datenverbindung“ geöffnet und wieder geschlossen, vgl. Abbildung 2. Dieses Konzept vereinigt die Anforderung an effiziente Nutzung des Netzes mit der Anforderung einer zuverlässigen Dateiübertragung.

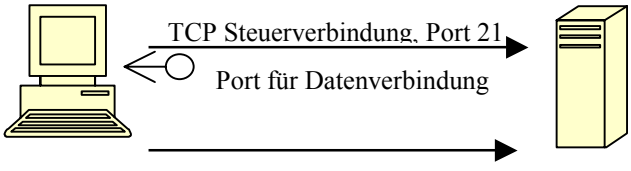


Abbildung 2: **Steuer und Datenverbindung in FTP**

Da das “Distributed Proxy” Pattern [SRG 02], [SRGA 02] keine Verbindung über die gesamte Sitzung zulässt, ist das Internet FTP Protokoll [RFC 959] mit diesem Pattern nicht implementierbar.

Ein praxisrelevantes Entwurfsmuster, das als allgemeine Basis zum Entwurf von Kommunikationsprotokollen dient, darf den Entwurf existenter, bereits bewährter Protokolle nicht unmöglich machen. Gleichzeitig sollte es vom wissenschaftlichen Anspruch her ein Maximum an Unabhängigkeit der einzelnen Schichten bewahren.

Darum muss das Distributed Proxy Pattern verallgemeinert werden um als Basis für den Entwurf und die Implementierung beliebiger praxisrelevanter Kommunikationsprotokolle zu dienen.

Die Neutralität bezüglich des Kommunikationsprotokolls ist ein wichtiges Qualitätsmerkmal eines Musters für verteilte Systeme.

In diesem Artikel wird das “Distributed Proxy” Pattern [SRG 02], [SRGA 02] so verallgemeinert, dass es dieser Anforderung genügt und wird entsprechend als *„General Distributed Proxy”* bezeichnet.

Dieses *„General Distributed Proxy”* Pattern beeinflusst Anzahl und Dauer der vom Kommunikationsprotokoll erzeugten Verbindungen im Gegensatz zum “Distributed Proxy” Pattern [SRG 02], [SRGA 02] nicht.

Das General Distributed Proxy Pattern umfasst die selben drei Schichten wie das ursprüngliche Distributed Proxy und bewahrt die Unabhängigkeit der einzelnen Schichten weitgehend. vgl. Abbildung 3.

| Anwendungsschicht: Anwendungsfunktionen |
|---|
| Logische Schicht: Verbindungsverwaltung **starr** |
| Physikalische Schicht Datenübertragung |

a. Distributed Proxy

| Anwendungsschicht: Anwendungsfunktionen |
|---|
| Logische Schicht: Verbindungsverwaltung **flexibel** |
| Physikalische Schicht Datenübertragung |

b. General Distributed Proxy

Abbildung 3: **Die Schichten**

Der Einfachheit der Darstellung wegen wird es hier mit ausschließlich passiven Verbindungen (Client öffnet Verbindung zum Server) dargestellt. Eine Erweiterung auf aktive Verbindungen ist jedoch analog zu [BRS 00] S. 285 ff sehr einfach.

Darüber hinaus wird das Dispatcher Pattern [BMR 96] S 324 ff nicht eingebunden. Dieses lässt sich in derselben Weise einbinden wie in [BRS 00] S. 285 ff. Dabei sollte der Dispatcher in der Version *„Client-Dispatcher-Server-with-Communication-Managed-by-Client“* [BMR 96] S 332 eingebunden werden, da das

Verbindungsmanagement im Distributed Proxy Pattern Aufgabe der Proxy Objekte ist. Man spricht dann besser von *„Communication-managed-by-Proxy.*

Nach einer kurzen Begriffsbestimmung eines Kommunikationsprotokolls (Kap. 2) wird ein Protokollentwurf exemplarisch durchgeführt. Das *„General Distributed Proxy"* Pattern wird dabei vorgestellt (Kap. 3) und mit dem *Distributed Proxy"* Pattern verglichen.

*„(General) Distributed Proxy"* bezeichnet im Folgenden immer beide Muster bei Aussagen, die für beide Entwurfsmuster gelten.

Als Beispiel wird ein vereinfachtes File Transfer Protokolls gewählt, das die für den Protokollentwurf wesentlichen Teile des passiven FTP [RFC 959] bewahrt.

So wird gezeigt, das General Distributed Proxy ist eine geeignete Basis für den Entwurf eines praxisrelevanten Kommunikationsprotokolls.

Im Anschluss daran wird kurz ein zum General Distributed Proxy alternatives Entwurfsmuster diskutiert (Kap. 4) und die beim Entwurf eines Kommunikationsprotokolls zu treffenden Entscheidungen werden aufgelistet (Kap. 5).

### 1.2 Diskussion weiterer Methoden zum Entwurf eines Kommunikationsprotokolls

Diese Papier beschränkt sich auf UML als Entwurfs – Sprache, da mit dem kombinierten Einssatz verschiedener Sprachen jedes mal ein Notationswechsel verbunden, was der Idee von UML widerspricht, vgl. auch [KRÄ 99] S. 15.

Der asynchrone Byte Stream, der die Prozesse verbindet, wird hier *nicht* als Objekt dargestellt, ähnlich [BMR 96] S. 313 - 432, anders als in [KRÄ 99] S. 56 , 57, 81, da er für die Klienten von Forwarder und Receiver transparent ist.

Diese sehen nur die Forwarder und Receiver Objekte sowie deren Methoden, „send( )" bzw. „receive( ) ".

Um einen Notationswechsel zu vermeiden werden insbesondere auch SDL, Leslie Lamports *Temporal Logic Actions(TLA)* nach Herrmann [HER 99] *Message Sequence Charts (MSC) [Z.120]* und die erweiterten *Object MSC (OMSC)* [BMR 96] nicht eingesetzt.

Der Implementierungsnähe wegen ist SDL ohnehin nur dann sinnvoll, wenn ein Generator zur Erzeugung von Source Code eingesetzt wird.

Die formale Beschreibungssprache Leslie Lamports *Temporal Logic Actions(TLA)* baut auf der Notation der Prädikatenlogik auf und ist darum für Entwickler ohne entsprechende Vorkenntnisse schwer erlernbar. Der Lernaufwand ist für einen nur gelegentlichen Einsatz nicht vertretbar.

Für die Beschreibung von Szenarien ist der Einsatz *Object MSC (OMSC)* grundsätzlich empfehlenswert, da Object MSCs mächtiger sind als Sequenz Diagramme von UML.
Dennoch beschränkt sich dieser Artikel auf den Einsatz von UML und damit auf *„Sequence Diagrams"*, eine Vereinfachung der MSCs.

## 2 Was ist ein Kommunikationsprotokoll?

Ein Kommunikationsprotokoll definiert die Regeln für den Nachrichtenaustausch zwischen Objekten verschiedener Prozesse einer verteilten Anwendung auf der Basis des benutzten Interprozess-Kommunikationsmechanismus (IPC).

Der Entwurf eines Kommunikationsprotokolls weist verschiedene Freiheitsgrade auf, d. h. verschiedene Entscheidungen müssen getroffen werden. Insbesondere müssen die Vor- und Nachteile einer verbindungsorientierten und einer verbindungslosen Kommunikation abgewogen werden.

Als Beispiel wird ein für die Belange dieser Arbeit einfaches FTP Protokoll entwickelt. Diese Anwendung hat eine hohe Anforderung an die Zuverlässigkeit des Dateitransfers, insbesondere an die Reihenfolge und Vollständigkeit der Bytes einer transferierten Datei. Darum wird hier ausschließlich verbindungsorientierte Kommunikation betrachtet.

Ferner wird TCP als IPC benutzt. Dies stellt keine prinzipielle Einschränkung dar.

TCP stellt zwei Prozessen einen bidirektionalen Byte Stream für die Kommunikation zur Verfügung. Dafür benötigen beide Prozesse eine TCP Adresse:

- Rechnername oder IP Adresse und
- Port – Nummer.

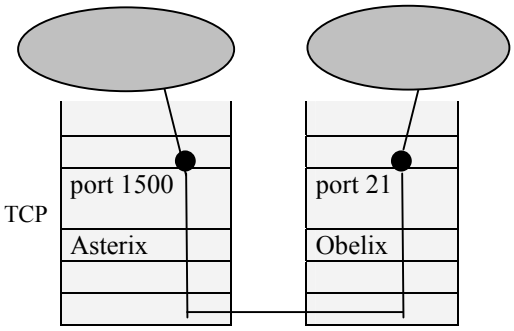


Abbildung 4: **Client hat Verbindung zum Server**

Die folgende Tabelle 1 beschreibt die für eine erfolgreiche Kommunikation eindeutig festzulegenden Regeln und Rahmenbedingungen. Diese werden entsprechend dem (General) Distributed Proxy Pattern in drei Schichten gegliedert. Ferner wird jede Regel mit der zwischenmenschlichen Kommunikation verglichen.

| Schicht | Zweck der Regel | Gegenstand der Regel | Übertragung auf zwischenmenschlichen Kommunikation |
|---|---|---|---|
| **funktional** | Kommunikations-gegenstand | Welche Inhalte werden übertragen. | Welche Inhalte werden kommuniziert, z.B. die einer Tagesordnung. |
| | Kommunikations-einschränkungen (Constraints) | Welcher Prozess beginnt die Kommunikation?<br>Welche Nachricht darf wann gesendet werden?<br>Welche Nachricht darf wie beantwortet werden?<br>Welche „Inhalte“ kann eine Nachricht haben? | Wer beginnt das Gespräch?<br>Ein Gespräch beginnt mir einer Begrüßung und endet mit einer Verabschiedung.<br>Einer Frage nach der Uhrzeit sollte man nicht mit einem Hinweis auf das Wetter beantworten.<br>Ein Gespräch hat einen Gesprächsgegenstand |
| **logisch** | Definition und Zuordnung von Verbindungen | Welche Verbindungen (Kommunikationskanäle) gibt es und wofür werden sie benutzt? | Sprache und Graphik.<br>Wofür benutzt man Sprache und wofür besser Graphik? |
| | Verbindungsauf- und -abbau | Welcher Prozess baut wann welche Verbindung auf? | Wer geht zu wem?<br>Der Berg zum Propheten! |
| | Adressinformation, Adressierung | Wie wird Adressinformation ausgetauscht?<br>Welche Adressierungsmethode wird benutzt? | Wie erfährt die eine Person, wo die andere ist?<br>Adresse = Land, Stadt, Straße, Hausnummer, Zimmernummer. |
| | IPC | Welcher IPC wird benutzt? | Z. B. Sprache: Luft,<br>Graphik: Papier |
| **physi-kalisch** | Objekt Repräsentation einer Nachricht | Wie kann man die Nachricht als Objekt repräsentieren? | Welche Sprache benutzen die Gesprächspartner?<br>Z. B. Deutsch., UML |
| | Byte Repräsentation einer Nachricht | Welchem Bytemuster (marshaling) muss eine Nachricht genügen? | Wie wird die Nachricht codiert?<br>Z. B. mit Schallwellen., Bleistift |

Tabelle 1: **Kommunikations-Regeln und –Rahmenbedingungen**

## 3 Protokollentwurf mit dem General Distributed Proxy Pattern

Da das passive FTP Protokoll einen flexiblen Verbindungsauf- und –abbau benötigt, ist es ein gutes Beispiel um die Flexibilität einer Methode zum Entwurf von Kommunikationsprotokollen zu demonstrieren.

Das FTP Protokoll wird hier vereinfacht und auf die für den flexiblen Verbindungsauf- und -abbau notwendigen Funktionen reduziert, vgl Abbildung 5: Use Case Diagramm des vereinfachten FTP Protokolls.

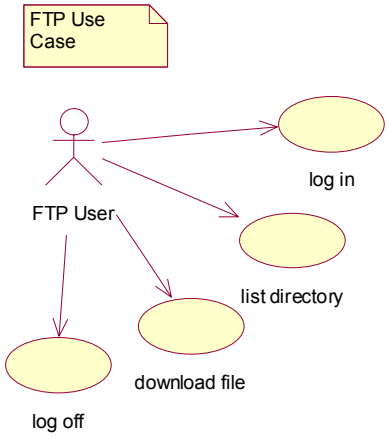


Abbildung 5: **Use Case Diagramm vereinfachtes FTP Protokoll**

Ein Anwendungsprotokoll sollte implementierungsgerecht definiert werden. Da die Implementierung i. a. in mehreren Schichten erfolgt, ist auch das Protokoll entsprechend der Entwurfsmuster-Schichten zu definieren.

Im Folgenden wird das General Distributed Proxy Pattern als Basis für die schichtweise Definition eines Kommunikationsprotokolls

vorgestellt. Das wird am Beispiel eines vereinfachtes FTP Protokolls demonstriert.

Das (General) Distributed Proxy Pattern beinhaltet drei Schichten: *die funktionale, logische* und *physikalische Schicht.*

Letztere wird durch die betrachteten Muster nochmals aufgeteilt: die *Obere Teilschicht mit Connector – Acceptor, Communicator* und die *untere Teilschicht mit Forwarder – Receiver.*

..Jede Schicht besteht aus mehreren Klassen bzw. deren Objekten. Da die jeweils obere Schicht die darunter liegende benutzt, stellt sie die Anforderungen an die untere Schicht. Diese Anforderungen legen die Methoden der Klassen der unteren Schicht oft weitgehend fest.

## 3.1 Die funktionale Schicht

### 3.1.1 Die Aufgaben der funktionalen Schicht

Die funktionale Schicht beinhaltet den Kommunikationsgegenstand und die Kommunikationseinschränkungen (Constraints), vgl. Tabelle 1: Kommunikations- Regeln und – Rahmenbedingungen.

Sie legt die Methoden fest, die der Server dem Client bietet.

### 3.1.2 Die Klassen der funktionalen Schicht

Die Klassen der funktionalen Schicht im (General) Distributed Proxy Pattern sind *„Client“* und *„Server“.*

Diese Schicht wird mit Hilfe der Client und Server Objekte implementiert.

### 3.1.3 Das Kommunikationsprotokoll der funktionalen Schicht

In diese Schicht werden der Kommunikationsgegenstand und Kommunikationseinschränkungen festgelegt. Das geschieht durch die Definition der Methoden der Server Klasse und deren Constraints, vgl. Beispiel vereinfachtes FTP unten.

**Beispiel vereinfachtes FTP:**

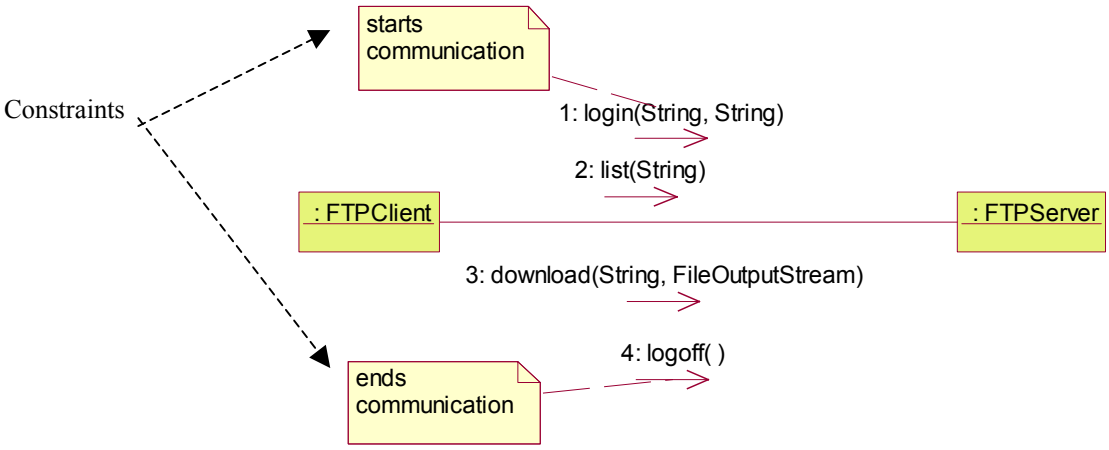


Abbildung 6: **Das Client Objekt benutzt die Methoden des Server Objekts**

Die vereinfachte FTP Anwendung umfasst nur die Funktionen „login“, „Verzeichnis anzeigen“, „Datei herunterladen“ und „Sitzung beenden“.

Darum hat die Server Klasse folgende Methoden:

- void login (String userID, String passWord): startet eine FTP Sitzung und hat als Parameter die user ID und das Passwort des Benutzers. Falls das login nicht O.K. ist, wird eine Exception geworfen.
- String[ ] list (String dirName): hat als Parameter einen Pfad / Name eines Verzeichnisses relativ zum aktuellen Verzeichnis und gibt die Liste der darin enthaltenen Dateien zurück. Falls zuvor kein erfolgreiches „log in“ gerufen wurde, wird eine Exception geworfen.
- void download (String fromFilename, FileOutputStream toFile): kopiert die Datei mit relativem Pfad / Namen fromFilename in die zu dem übergebenen FileInputStream gehörende Datei. Falls zuvor kein erfolgreiches „log in“ gerufen wurde, wird eine Exception geworfen.
- void logoff( ): beendet die FTP Sitzung.

**Ende Beispiel vereinfachtes FTP**

### 3.1.4 Vergleich der funktionalen Schicht der beiden Pattern

Die beiden Entwurfsmuster unterscheiden sich in der Behandlung einer Sitzung, vgl. Tabelle 2: die funktionale Schicht.

| **Schritt** | **beide Pattern gleich** | | |
|---|---|---|---|
| | **Distributed Proxy** | | **General Distributed Proxy** |

| | | | |
|---|---|---|---|
| 1. | Das Client Objekt sendet dem Server Objekt Nachrichten, d.h. es benutzt die Methoden des Server Objekts. | | |
| 2 | Jede Nachricht ist eine selbständige Kommunikation..<br>Nur der Client Prozess kennt den Anfang und das Ende einer Sitzung. | | Falls es eine Verbindung über die gesamte Sitzungsdauer gibt, zeigt das Client Objekt dem Server Objekt das Ende der Sitzung mit einer geeigneten Nachricht an. |

Tabelle 2: **die funktionale Schicht**

### 3.1.5 Unterstützung der inkrementellen Implementierung durch die funktionale Schicht

Falls man die Implementierung der funktionale Schicht, Client und Server Objekt, in *einen* Prozess integriert erhält man eine lokale Anwendung mit der Funktionalität der geplanten verteilten Anwendung.

Die hier vorgestellte Methodik erlaubt daher eine inkrementelle Implementierung, d. h. zuerst wird die funktionale Schicht implementiert und als lokale Anwendung getestet. Sobald die funktionale Schicht als lokale Anwendung die funktionalen Anforderungen erfüllt, wird Sie zu einer verteilten Anwendung erweitert.

Bei einer verteilten Anwendung gehören die Client und Server Objekte verschiedenen Prozessen an. Der Client kann somit die Servermethoden nicht mehr direkt aufrufen. Client und Server benutzen darum die logische Schicht zum Nachrichtenaustausch.

## 3.2 Die logische Schicht

Im General Distributed Proxy Pattern wird die logische Schicht gegenüber dem Distributed Proxy Pattern erweitert. Sie ist der Kern der Erweiterung.

### 3.2.1 Aufgaben der logischen Schicht

Diese Schicht bildet die Brücke zwischen der funktionalen und der physikalische Schicht.
Sie ist zuständig für
- Definition und Zuordnung von Verbindungen
- Verbindungsauf- und -abbau
- Austausch von Adressinformation und Adressierung
- IPC Nutzung ,

vgl. Tabelle 1: „Kommunikations-Regeln und –Rahmenbedingungen.

### 3.2.2 Die Klassen der logischen Schicht

Im (General) Distributed Proxy Pattern besteht die logische Schicht aus einem Proxy Klassen Paar: *„ClientSideProxy – ServerSideProxy.*

Die logische Schicht wird mit einem *„ClientSideProxy“* Objekt und einem *„ServerSideProxy“* Objekt implementiert.

Das ClientSideProxy Objekt vertritt das Server Objekt im Adressraum des Client Prozesses und das ServerSideProxy Objekt vertritt das Client Objekt im Adressraum des Server Prozesses, vgl. Abbildung 7. Die Proxy Objekte vertreten das Partner Objekts.

**Beispiel vereinfachtes FTP**

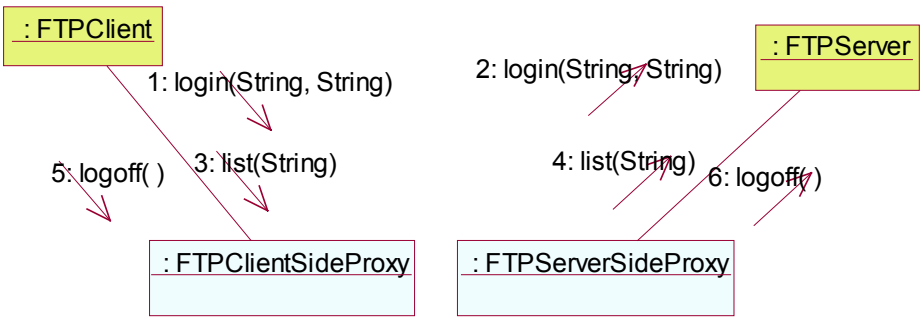


Abbildung 7: **Die Proxy Objekte vertreten das Partner Objekts**

**Ende Beispiel vereinfachtes FTP**

Entsprechend den Anforderungen der funktionalen Schicht ist die ClientSideProxy Klassen vom Server Reference Interfaces abgeleitet. Auch die ServerSideProxy Klasse benötigt im Wesentlichen die Methoden der FTPServer Klasse. Die Proxy Klassen implementieren diese Methoden *entsprechend dem Kommunikationsprotokoll.*

Um Verbindungsinformation austauschen zu können, ist ggf. ein Änderung der Methodensyntax auf Seiten der ServerSideProxy Klasse notwendig.

Wie das Beispiel zeigt wird, benötigt die ServerSideProxy Klasse eine geänderte Methode („download“) und auch weitere Methoden zur Implementierung des Protokolls.

Darum wurde das Reference Interface aufgeteilt vgl. Abbildung 12: Die Klassen des Beispiels FTP.

### 3.2.3 Das Kommunikationsprotokoll der logischen Schicht

Die logische Schicht eines Kommunikationsprotokoll legt fest

- *welche Verbindungen es gibt,*
- *wozu welche Verbindung benutzt wird,*
- *welcher Prozess welche Verbindung aufbaut - und wann sie abgebaut wird,*
- *wie Adressinformation ausgetauscht wird,*
- *welche Adressierungsmethode benutzt wird und*
- *welcher IPC.*

Anzahl, Dauer und Benutzung der Verbindungen müssen der Funktion der Anwendung angepasst sein.

Passive Verbindungen (von Client zum Server) sind im Zusammenhang mit der Nutzung einer Firewall i. a. aktiven (vom Server zum Client) vorzuziehen.

Die Anwendung kann die Adressierung des benutzten IPCs übernehmen oder ihre eigene Adressierung definieren. Der zweite Fall macht die Anwendung unabhängig vom benutzten IPC Mechanismus. Dieser kann dann bei Bedarf leicht ausgetauscht werden.

Andererseits erfordert eine selbst definierte Adressierung immer eine Übersetzung der anwendungsspezifischen Adresse in die Adresse des IPC Mechanismus.

Die folgende Tabelle gibt einen Überblick über die Vor – und Nachteile einer anwendungsspezifischen Adresse.

| **IPC Adressierung** | **eigene Adressierung** |
|---|---|
| + einheitlich mit anderen Internet – Anwendungen | + unabhängig vom benutzten IPC Mechanismus |
| + keine Übersetzung in IPC Adresse | |

Tabelle 3: **Vergleich einer IPC Adressierung mit einer eigenen Adressierung durch die Anwendung**

**Anmerkung:** Ein Kommunikationsprotokoll legt prinzipiell nicht fest, ob ein Client von einem parallelen oder einem sequentiellen Server bedient wird. Bei der Implementierung der Klassen des General Distributed Proxy Pattern kann man dies jedoch integrieren.

Für das Beispiel der vereinfachten FTP Anwendung ist es sinnvoll, dass ein Server mehrere Clients gleichzeitig bedient. Darum wird die Klasse ServerSideCommunicator, vgl. unten hier von der Thread Klasse abgeleitet.

Außerdem werden die Aufgaben der ServerSideProxy Klasse auf zwei Proxy Klassen aufgeteilt: Die ServerSideProxyInit und die FTPServerSideProxy Klassen.

Alternativ zu Threads kann man auch das „Reactor“ Pattern [BRS 00] S 179 benutzen.

**Beispiel vereinfachtes FTP:**

- **welche Verbindungen gib es?**
- **Wozu wird welche benutzt?**
- **Welcher Prozess baut sie auf, welcher Prozess wartet, wann wird sie abgebaut?**
- **Wie wird Adressinformation ausgetauscht?**

Im passiven FTP Protokoll baut der Client Prozess eine Steuerverbindung zum Server Prozess für die gesamte Sitzungszeit auf Ständiger Auf- und Abbau von Verbindungen würde das Netz zu stark belasten. Auf diese Steuerverbindung wartet der Server an einem „well known“ Port.

Zu übertragende Dateidaten dürfen keinesfalls mit Steuerinformation vermischt werden..

Um die Fehlertoleranz der Anwendung zu vergrößern und um Dateitransfer im Hintergrund zu ermöglichen baut in einer FTP Anwendung der Client Prozess für jeden Dateitransfer zusätzlich zur Steuerverbindung eine eigene Datenverbindung auf. Nach dem Dateitransfer wird diese wieder abgebaut.

Zum Verbindungs-Aufbau teilt der Server Prozess dem Client Prozess eine Portnummer auf der Steuerverbindung mit, an der er die Datenverbindung erwartet. Dieser passive Modus erleichtert den Betrieb über eine Firewall.

**Welche Adressierungsmethode und welcher IPC wird benutzt?**

FTP ist eine OSI - internetfähige Anwendung, darum bietet sich die Internet - Adressierung an: Port Nummer und IP Adresse oder Port Nummer und Host Name. Dies ist bereits die TCP Adressierung.

Da bei der Datei-Übertragung Vollständigkeit und Reihenfolge der Daten wichtig ist, wird TCP mit der Socket Schnittstelle als IPC gewählt.

**Ende Beispiel vereinfachtes FTP**

### 3.2.4 Implementierung und Vergleich der logischen Schicht der beiden Pattern

Die beiden Pattern unterscheiden sich in der logischen Schicht.

Beim Distributed Proxy Pattern wird die Implementierung der Proxy Klassen großteils durch das Pattern vorgegeben. Diese Implementierung wiederum bestimmt das Kommunikationsprotokoll weitgehend.

Beim General Distributed Proxy Pattern wird zuerst das Kommunikationsprotokolls definiert. Dieses bestimmt weitgehend, wie die Methoden der ClientSide- und ServerSideProxy Klassen implementiert werden.

Die folgenden Tabellen beschreiben die Proxy Objekte im Einzelnen und vergleichen die beiden Entwurfsmuster.

**Initialisierung der logischen Schicht: ClientSideProxy**

| **Schritt** | **beide Pattern gleich** | | |
|---|---|---|---|
| | **Distributed Proxy** | | **General Distributed Proxy** |
| 1. | Das Client Objekt *benutzt* ein ClientSideProxy Objekt anstelle des Server Objekts.<br>**Anmerkung**: Im Beispiel vereinfachtes FTP wurde die Aufgabe der ServerSideProxy Klasse auf die Klassen ServerSideProxyInit und FTPServerSideProxy aufgeteilt. | | |
| 2. | | | Das ClientSideProxy Objekt erzeugt ggf. am Anfang der Sitzung *eine oder mehrere Verbindungen*, die während der *ganzen* Sitzung erhalten bleiben.<br>Die sinnvolle Anzahl der Verbindungen ist anwendungs-spezifisch und wird durch das Protokoll definiert, das die ClientSideProxy Klasse implementiert.<br>Im Falle des einfachen FTP Protokoll-Beispiels wird die Steuerverbindung aufgebaut, vgl. Abbildung 8.<br>Zum Verbindungsaufbau benutzt das ClientSideProxy Objekt ein *„Connector“* Objekt. der physikalischen Schicht.<br>*Anmerkung:* In Java kann hierzu direkt ein *Socket* Objekt benutzt werden. |
| 3. | | | DasClientSideProxy Objekt benutzt für den Zugriff auf jede seiner Verbindungen ein *ClientSideCommunicator*<br>Objekt der physikalischen Schicht, vgl. Abbildung 8. |

Tabelle 4: **Initialisierung der logischen Schicht des Client Prozesses**

**Initialisierung der logischen Schicht: ServerSideProxy**

| **Schritt** | **beide Pattern gleich** | | |
|---|---|---|---|
| | **Distributed Proxy** | | **General Distributed Proxy** |
| 1. | Der Server Prozess erzeugt ein ServerSideProxy Objekt und ein Server Objekt. | | |
| 2. | Der Server Prozess erzeugt ein *„Acceptor“* Objekte und dieses warten auf eine Verbindung eines Client Prozesses | | Der Server Prozess erzeugt ggf. in mehreren Threads für jeden Connector, den der Client Prozesses am Sitzungsanfang erzeugt, ein zugehöriges *„Acceptor“* Objekte. |
| 3. | Die Acceptor Objekte warten auf eine Verbindung<br>*Anmerkung:* In Java kann hierzu direkt ein *ServerSocket* Objekt benutzt werden. | | |
| 4. | Sobald eine Verbindung besteht, erzeugt der Server Prozess ein ServerSideCommunicator Objekt der physikalischen Schicht für diese Verbindung, vgl. Abbildung 8. | | |

Tabelle 5: **Initialisierung der logischen Schicht des Server Prozesses**

**Beispiel vereinfachtes FTP**

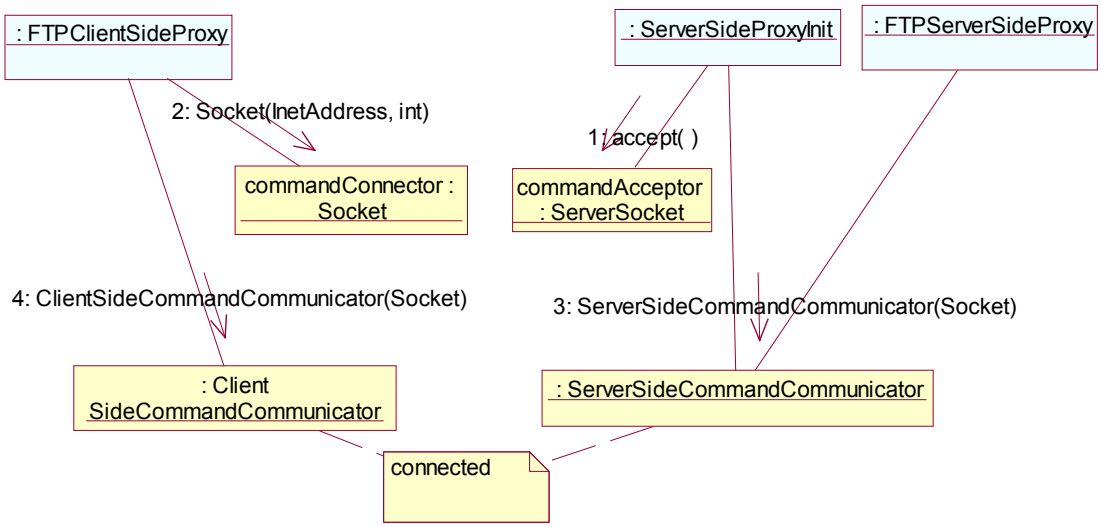


Abbildung 8: **Beispiel Initialisierung von Proxy Objekten mit einer Sitzungsverbindung**

**Ende Beispiel vereinfachtes FTP**

**Bearbeitung eines Aufrufs einer Server-Methode durch die logische Schicht: ClientSideProxy**

| **Schritt** | **beide Pattern gleich** | | |
|---|---|---|---|
| | **Distributed Proxy** | | **General Distributed Proxy** |
| | Das Client Objekt *benutzt* ein ClientSideProxy Objekt anstelle des Server Objekts. | | |
| 1. | Das ClientSideProxy Objekt nimmt den *nächsten Aufruf* einer seiner Methoden durch das Client Objekts entgegen: | | |
| 2. | Das ClientSideProxy Objekt benutzt ein *Connector* Objekt der physikalischen Schicht, um eine Verbindung zu erstellen.<br>Der Zugriff auf diese Verbindung erfolgt mit einem *ClientSideCommunicator* Objekt der physikalischen Schicht. | | In Abhängigkeit von der gerufenen Methode entscheidet das ClientSideProxy Objekt welche bestehende Verbindung (d.h. welches ClientSideCommunikator Objekt) benutzt wird oder ob eine neue Verbindung und ein neues ClientSideCommunicator Objekt erstellt wird.<br>Bevor die neue Verbindung erstellt werden kann, wird ggf. die Steuerleitung benutzt, um beispielsweise eine Port Nummer auszutauschen. Dazu benötigt die ServerSideProxy Klasse ein Interface, das von dem von der ClientSideProxy Klasse benutzten Interface abweicht.<br>Zum Erstellen einer Verbindung wird ein neues Connector Objekt benutzt, vgl. Abbildung 10: |
| 3. | Das ClientSideProxy Objekt reicht den Methodenaufruf des Client Objekts an seine physikalische Schicht weiter: Aufruf einer Methode des ClientSideCommunicator Objekts. vgl. Abbildung 9, 11. | | |
| 4. | Das ClientSideProxy Objekt leitet die *Rückgabewerte* an den Client weiter, . vgl. Abbildung 9, 11. | | |
| 5.. | Die Verbindung wird geschlossen | | Falls eine neue Verbindung für den Methodenaufruf erstellt wurde, wird die Verbindung geschlossen. |
| 6. | Weiter bei 1. bis die vom ClientSideProxy empfangene Methode das Ende der Sitzung anzeigt. | | |

Tabelle 6: **das ClientSideProxyObjekt nimmt einen Methodenaufruf an**

**Bearbeitung eines Aufrufs einer Server-Methode durch die logische Schicht: ServerSideProxy**

| **Schritt** | **beide Pattern gleich** | | |
|---|---|---|---|
| | **Distributed Proxy** | | **General Distributed Proxy** |
| | Der Server Prozess benutzt ein ServerSideProxy Objekt anstelle des Client Objekts. | | |
| 1. | Das ServerSideCommunicator Objekt (physikalische Schicht) liest die vom Client aufgerufenen Methode und gibt sie and das ServerSideProxy Objekt weiter, . vgl. Abbildung 9, 11. | | |
| 2. | | | Falls der Methodenaufruf eine neue Verbindung erfordert, wird diese mit einem Acceptor Objekt erwartet und dann das zugehörige ServerSideCommunicator Objekt vom Server Prozess erstellt, . vgl. Abbildung 9, 11.<br>Ggf. muss zuvor beispielsweise eine Port Nummer auf der Steuerverbindung an den ClientSideProxy übermittelt werden. |

| | |
|---|---|
| 3. | Das Server-Side Proxy Objekt leitet den Methodenaufruf an das Server Objekt weiter, vgl. Abbildung 9, 11. |
| 4. | Das Server-Side Proxy Objekt wartet auf die Rückgabewerte vom Server Objekt, um sie der physikalischen Schicht zu übergeben, vgl. Abbildung 9, 11. |

Tabelle 7: **das ServerSideProxy Objekt leitet einen Methodenaufruf weiter**

**Anmerkung:**
Jede Verbindung hat ihre eigene physikalische Schicht, d. h. für verschiedene Verbindungen werden verschiedene Connector und Acceptor Objekte sowie verschiedene ClientSide- und ServerSideCommunicator Objekte benutzt.
..Ob dies jeweils Objekte einer Connector, Acceptor, ClientSide- und ServerSideCommunicator Klasse sind oder ob es verschiedene solche Klassen gibt, ist anwendungsabhängig, vgl. physikalische Schicht, unten.

**Beispiel vereinfachtes FTP**

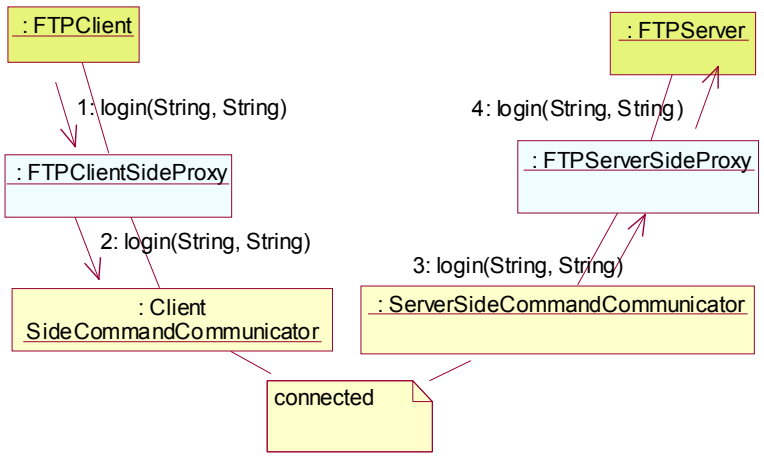


Abbildung 9: **Beispiel für einen Methodenaufruf der nur die Steuerverbindung benötigt**

Bei einem passiven FTP wird für jede zu übertragende Datei eine Datenverbindung vom Client Prozess zum Server Prozess aufgebaut und die Datei wird auf dieser Verbindung übertragen. Dies ist Aufgabe der Proxy Objekte und der von ihnen benutzten *„dataConnector“* und *„dataAcceptor“* Objekte, vgl. Abbildung 8.

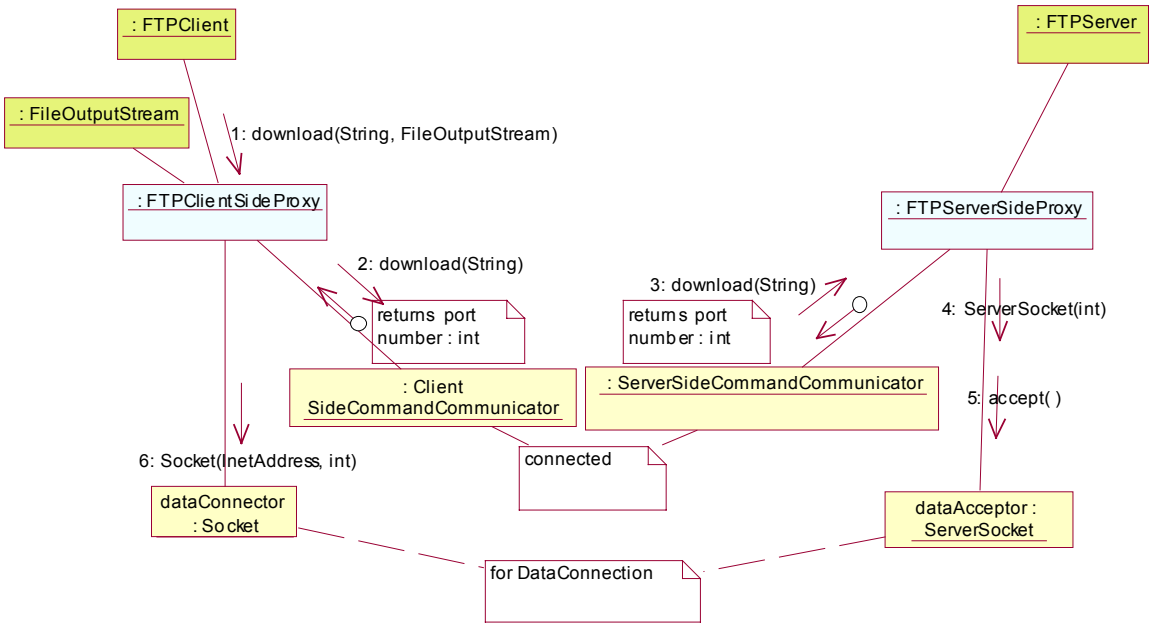


Abbildung 10: **Beispiel für einen Methodenaufruf der die Datenverbindung benötigt: Verbindungsaufbau**

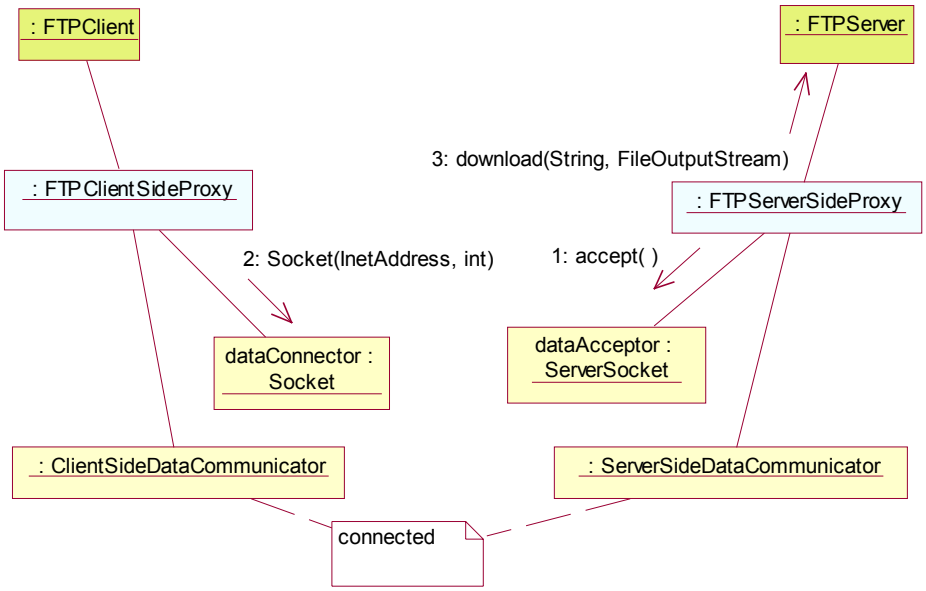


Abbildung 11: **Beispiel für einen Methodenaufruf der die Datenverbindung benötigt: Datenübertragung Teil 1**

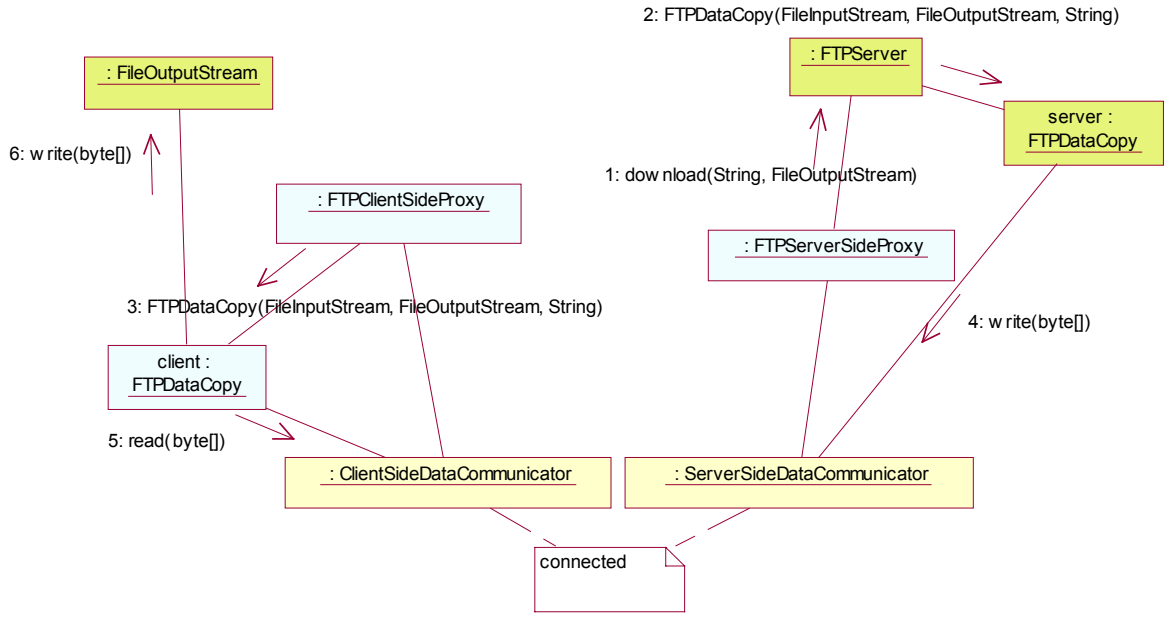


Abbildung 12: **Beispiel für einen Methodenaufruf der die Datenverbindung benötigt: Datenübertragung Teil2**
**Ende Beispiel vereinfachtes FTP**

### 3.2.5 Unterstützung der inkrementellen Implementierung durch die logische Schicht

Falls die Anwendung im Sinne der inkrementellen Implementierung um neue Funktionen erweitert wird, kann das Kommunikationsprotokoll auf der Basis des General Distributed Proxy Pattern leicht auch um neue Verbindungen für die neuen Funktionen erweitert werden.

### 3.2.6 Anforderungen der logischen an die physikalische Schicht

Im (General) Distributed Proxy Pattern wird für jede Verbindung eine physikalische Schicht „zwischen“ die Proxy Objekte geschoben. Sie unterstützt die Verbindungsverwaltung.

Für den Verbindungsaufbau benutzen die Proxy Objekte ein Connector – Acceptor Klassen Paar. Dafür kann man die Java Klassen Socket – ServerSocket benutzen oder nachempfinden,

falls man eine andere Programmiersprache einsetzt.

Zum Nachrichtenaustausch über die IPC Verbindung benutzen die Proxy Objekte ein ClientSideCommunicator – ServersideCommunicator Objekt Paar [SRG 02]

### 3.3 Die physikalische Schicht

Die physikalische Schicht ist im General Distributed Proxy Pattern und dem Distributed Proxy Pattern gleich. Sie wird mit *„Acceptor - Connector“, Communicator“* und *„Forwarder - Receiver“* realisiert

## 4 Zusammenfassung

Anhand eines vereinfachten FTP Protokolls konnte gezeigt werden, dass die Entwurfsmuster *“General Distributed Proxy”* , *„Acceptor - Connector“, Communicator“* und *„Forwarder - Receiver“* den Entwicklungsprozess eines Kommunikationsprotokolls schichtweise gliedern, ohne das Protokoll selbst zu beeinflussen.

Die Entwurfsentscheidungen werden schichtweise getroffen und das Protokoll wird mit Hilfe der Klassenspezifikation (Syntax und Semantik) der Schicht dokumentiert und realisiert.

Dafür wurde das *„Distributed Proxy”* Pattern in der logischen Schicht geeignet erweitert.

Zusätzlich unterstützt das *General Distributed Proxy* Muster inkrementelle Implementierung.

Diese Prozessverbesserung leistet zusammen mit der objektorientierte Methodik und der zugehörigen Beschreibungstechnik *UML* [OMG UML 1.3], wichtigen Beitrag zur Entwicklung von zuverlässigen und wartungsfreundlichen verteilten Systemen.

## 5 Dank

Meinem Kollegen Herrn Prof. Dr. K. Theobald gilt mein besonderer Dank für wertvolle Diskussionen.

## 6 Literatur

[BMR 96] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Siemens AG Deutschland: „Pattern - orientierte Softwarearchitektur Ein Pattern-System“ Addison-Wesley München, Boston, 1996

[BRS 00] Douglaus Schnidt, Michael Stal, Hans Rohnert, Frank Buschmann: „Pattern - orientierte Softwarearchitektur Vol 2, Pattern for concurrent and Network Objects“ John Wiley Chinchester, New York, ...2000

[HER 99] Peter Herrmann: „Problemnaher korrektheitssichernder Entwurf von Hochleistungsprotokollen“ Deutscher Universitätsverlag, Wiesbaden 1998 ISBN: 3-8244-2101-1

[KRÄ 99] M. J. Kräß: “Eine Integration formaler und objektorientierter Methoden bei der Entwicklung von Kommunikationsprotokollen“ Dissertation Fakultät für Elektrotechnik und Informationstechnik, TU München, 1999

[SIE 96] G. Siegmund: „Technik der Netze“ R. v. Decker´s Heidelberg 1996, ISBN: 3-7685-2495-7

[SRG 02] A. R. Silva, F. A.Rosa, T Goncalves: „Distibuted Proxy: A Design Pattern for Distributed Object Communication“ INESC/IST Technical University of Lisbon, 1000 Lisboa, Portugal www.cs.uiuc.edu download 09. 07. 02

[SRGA 02] A. R. Silva, F. A. Rosa, T Goncalves, Miguel Antunes: „Distibuted Proxy: A Design Pattern for Incremental Development of Distributed Applications“ INESC/IST Technical University of Lisbon, 1000 Lisboa, Portugal HKS – Hibbitt, Karlsson & Sorensen, Inc. Pawtucket, RI 02860, USA www.esw.inesc.pt download 09. 07. 02

[OMG: UML 1.3]OMG Unified Modeling Language Specification, Version 1.3, March 2000, download from: http//www.omg.org/technology/documents/formal/uml.htm: item: formal/2001-9-67. Link found by http//www.omg.org/uml/

[RFC 959] J. Postal and J. Reynolds :”File Transfer Protocol (FTP)”, RFC 959, Okt. 1985 www.rfc-editor.org/rfc/rfc959.txt

[Z.100 (3/93)] CCITT Specification and Design Language (SDL), ITU-T 1994

[Z.120 (11/1999)] Message Sequence Chart (MSC), ITU-T November 1999