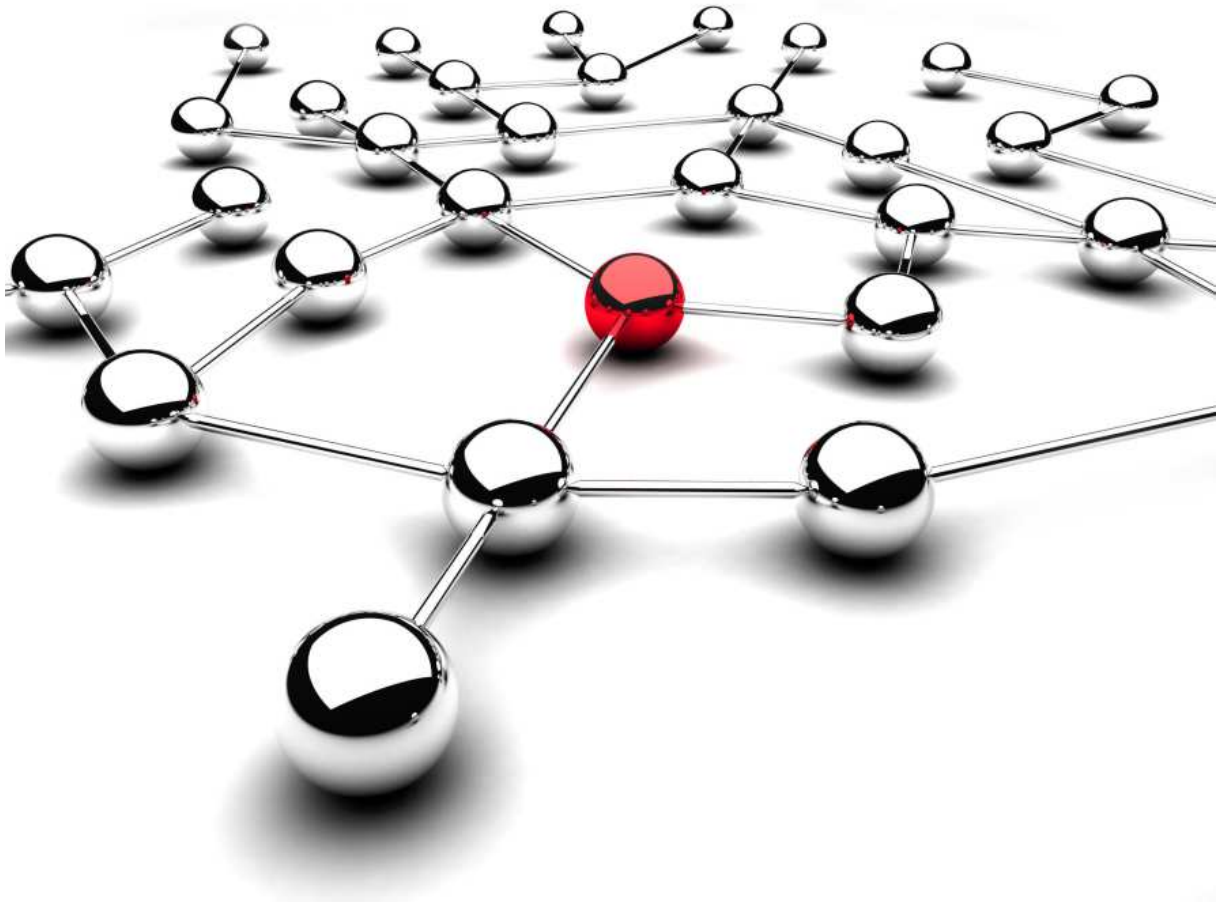


IRC Services in C

Diensteserver für einen Internet-Chat



Zürcher Hochschule für angewandte Wissenschaften
Betreuungsperson: Karl Brodowsky, Fachdozent für Systemprogrammierung
Erscheinungsjahr: 2015

Eine Semesterarbeit von Severin A. Müller

Abstract

Die vorliegende Arbeit behandelt die Entwicklung eines Dienste-Server für einen Text-Chat basierend auf dem Internet-Relay Chat (IRC) Protokoll. Die Einleitung schildert die Ausgangslage, die Motivation des Autors und die Ziele dieser Arbeit. Im zweiten Kapitel folgt ein Exkurs über die Geschichte des Internet Relay Chat und die Einführung in dessen Funktionsweise und das Protokoll. Darauf folgt in einer Marktanalyse eine Übersicht bestehender Lösungen sowohl für den reinen Chat als auch für Dienste oder gar integrierte Lösungen. Im vierten Kapitel folgt eine ausführliche Anforderungsanalyse in welcher jede Funktion die das zu entwickelnde Programm unterstützen soll detailliert beschrieben wird. Dazu wurde jeweils ein entsprechender Anwendungsfall erstellt und die Architektur in einer Grafik anschaulich dargestellt. Im darauffolgenden Kapitel wird auf die Umsetzung eingegangen. Da viele Code-Abschnitte sich wiederholen wurde jeweils das erwähnt, was für diese Arbeit von Interesse ist. Speziell auf die Aspekte der Systemprogrammierung (Signale, Timer) wird ein Augenmerk gelegt. Zudem bietet das einen Überblick über die Anwendung der Datenbank SQLite. Im Kapitel über das Testing findet sich ein kurzes Testing-Konzept sowie eine Anleitung wie das Programm korrekt aufgesetzt und betrieben werden. Das entsprechende Benutzerhandbuch ist Teil des Programms und wird in einem Abschnitt kurz beschrieben. Die Arbeit schliesst mit einem Ausblick auf zukünftige Funktionen und ein Fazit.

Der Autor konnte eine funktionierende Lösung in der vorgegebenen Zeit entwerfen und implementieren. Die Arbeit war sicherlich umfangreich, aber dank der Motivation des Autors wurden die Ziele dieser Arbeit erreicht.

Inhaltsverzeichnis

1 Einleitung	9
1.1 Themenwahl	9
1.2 Ziele der Arbeit.....	10
1.3 Aufgabenstellung	11
2 Einführung in IRC und Services	12
2.1 Geschichte des IRC.....	12
2.2 Funktionsprinzip des IRC.....	12
2.3 Protokoll	13
2.3.1 Einführung.....	13
2.3.1.1 Grundlagen	13
2.3.1.2 Operatoren	14
2.3.1.3 Channels	14
2.3.2 Die IRC Spezifikation	14
2.3.3 IRC Konzepte	14
2.3.3.1 1:1 Kommunikation	15
2.3.3.2 1:n Kommunikation	15
2.3.3.3 1:Alle Kommunikation	15
2.3.3 Nachrichten Details.....	15
2.3.3.1 Pass Nachricht	15
2.3.3.2 Nick Nachricht	15
2.3.3.3 User Nachricht.....	16
2.3.3.4 Server Nachricht.....	17
2.3.3.5 PRIVMSG.....	17
2.3.3.6 NOTICE.....	17
2.4 Services	17
2.4.1 Einführung.....	17
2.4.2 Nickserv	18
2.4.3 Chanserv.....	18
2.4.4 Opserv.....	18
2.4.5 Botserv	18
2.4.6 Adminserv	19
2.4.7 Festlegung der Grenzen	19

3 Marktanalyse	20
3.1 Webmaster ConferenceRoom.....	20
3.2 Unreal IRCd	20
3.3 IRC Services	20
3.4. Anope Services	20
3.5 Auspice.....	21
3.6 Fazit.....	21
4 Anforderungen	22
4.1 Basis-Server	22
4.1.1 Use Case Basis Server (UC S-01).....	22
4.1.2 Anforderungen Basis-Server	22
4.1.2.1 IRC Server	22
4.1.2.2 Konfiguration Basis-Server	23
4.1.2.3 Starten und Stoppen des Servers	23
4.1.2.4 Validierung Server-Konfiguration.....	23
4.1.2.5 Daten von Datenbank laden.....	24
4.1.2.6 Verbindung zum IRC Server.....	24
4.1.2.7 Dienste starten und verbinden.....	24
4.1.2.8 Daten in regelmässigen Abständen speichern	24
4.1.2 Dienste allgemein	25
4.1.3 Datenbank.....	25
4.1.4 Logging	25
4.2 Nickserv.....	26
4.2.1 Use Case Nickserv (UC NS-01).....	26
4.2.2 Anforderungen Nickserv	26
4.2.3 Anforderung Nickserv Hauptfunktion.....	27
4.2.4 Konfiguration Nickserv.....	27
4.2.5 Anforderungen Nickserv Unterfunktionen	29
4.2.5.1 Anforderungen ACC.....	29
4.2.5.2 Anforderungen ACCESS	29
4.2.5.3 Anforderungen AUTH	30
4.2.5.4 Anforderungen DROP	30
4.2.5.4 Anforderungen GETPASS.....	30

4.2.5.5 Anforderungen GHOST	30
4.2.5.6 Anforderungen IDENTIFY.....	31
4.2.5.7 Anforderungen INFO	31
4.2.5.8 Anforderungen LIST	31
4.2.5.9 Anforderungen LISTCHANS.....	31
4.2.5.10 Anforderungen NOTIFY	32
4.2.5.11 Anforderungen REGISTER.....	32
4.2.5.12 Anforderungen RELEASE	32
4.2.5.13 Anforderungen SET.....	33
4.2.5.14 Anforderungen SETPASS.....	33
4.2.6 Weitere Nickserv Anforderungen	33
4.2.6.1 Identifikationstimer	33
4.3 Chanserv	34
4.3.1 Use Case Chanserv (UC CS-01).....	34
4.3.2 Anforderungen Chanserv	34
4.3.3 Anforderung Chanserv Hauptfunktion	35
4.3.4 Konfiguration Chanserv	35
4.3.5 Anforderungen Chanserv Unterfunktionen.....	37
4.3.5.1 ACC	37
4.3.5.2 Anforderungen AKICK.....	37
4.3.5.3 Anforderungen AOP	38
4.3.5.4 Anforderungen DE-/HALFOP	38
4.3.5.5 Anforderungen DE-/OP	39
4.3.5.6 Anforderungen DE-/VOICE	39
4.3.5.7 Anforderungen DROP	39
4.3.5.8 Anforderungen GETPASS.....	39
4.3.5.9 Anforderungen HOP	40
4.3.5.10 Anforderungen IDENTIFY.....	40
4.3.5.11 Anforderungen INVITE.....	40
4.3.5.12 Anforderungen LIST	41
4.3.5.13 Anforderungen MDEOP.....	41
4.3.5.14 Anforderungen MKICK.....	41
4.3.5.15 Anforderungen REGISTER.....	42
4.3.5.16 Anforderungen SET.....	42
4.3.5.17 Anforderungen SETPASS.....	43
4.3.5.18 Anforderungen SOP.....	43

4.3.5.19 Anforderungen UNBAN	43
4.3.5.20 Anforderungen UOP	44
4.3.5.21 Anforderungen VOP	44
4.4 Operserv.....	45
4.4.1 Use Case Operserv (UC OS-01)	45
4.4.2 Anforderungen Operserv	45
4.4.3 Anforderung Operserv Hauptfunktion.....	45
4.4.4 Konfiguration Operserv.....	45
4.4.5 Anforderungen Operserv Unterfunktionen	46
4.4.5.1 Anforderungen AKILL.....	46
4.4.5.2 Anforderungen CHATOPS.....	47
4.4.5.3 Anforderungen CHGOST.....	47
4.4.5.4 Anforderungen GLOBAL	47
4.4.5.5 Anforderungen KILL.....	47
4.4.5.6 Anforderungen LOCAL.....	48
4.4.5.7 Anforderungen OPER	48
4.4.5.8 Anforderungen Server-Bans und Nicksperrern	49
4.5 Botserv	49
4.5.1 Use Case Botserv (UC BS-01)	49
4.5.2 Anforderungen Botserv	50
4.5.3 Anforderung Botserv Hauptfunktion	50
4.5.4 Konfiguration Botserv	50
4.5.5 Anforderungen Botserv Unterfunktionen	51
4.5.5.1 Anforderungen ADD/DEL	51
4.5.5.2 Anforderungen DE-/HALFOP	51
4.5.5.3 Anforderungen DE-/VOICE	51
4.5.5.4 Anforderungen DE-/OP	52
4.5.5.5 Anforderungen GETPASS.....	52
4.5.5.6 Anforderungen IDENTIFY.....	52
4.5.5.7 Anforderungen INFO	52
4.5.5.8 Anforderungen KICK.....	53
4.5.5.9 Anforderungen LIST.....	53
4.5.5.10 Anforderungen KICK.....	53
4.5.5.11 Anforderungen SET.....	53
4.5.5.12 Anforderungen SETPASS.....	54

4.6 Adminserv	54
4.5.1 Use Case Adminserv (UC AS-01)	54
4.6.2 Anforderungen Adminserv.....	54
4.6.3 Anforderung Adminserv Hauptfunktion	54
4.6.4 Konfiguration Adminserv	54
4.6.5 Anforderungen Adminserv Unterfunktionen	55
4.6.5.1 Anforderungen SAVEDATA.....	55
4.6.5.2 Anforderungen SQUIT	55
5. Umsetzung	56
5.1 Design	56
5.1.1 Architektur	56
5.1.2 Datenbank-Design.....	57
5.2 Setup IRC Server	58
5.3. Implementierung Konfiguration	58
5.4. Implementierung Server-Verbindung	60
5.4.1 Verbindung Basis-Server	60
5.4.2 Dienste	61
5.4.3 Verarbeitung von Befehlen durch den Basis-Server.....	62
5.4.4 Benutzer- und Channelverwaltung	66
5.5 Implementierung Dienste	66
5.5.1 Allgemein	66
5.5.2 Nickserv	67
5.5.2.1 Registrierung Nickname	67
5.5.2.2 Löschen eine Nickname.....	68
5.5.2.3 Identifikation	68
5.5.2.4 Timer	69
5.5.2.5 Weitere Nickserv Funktionen.....	70
5.5.3 Chanserv.....	70
5.5.3.1 Operatoren-Listen	70
5.5.4 Opserv.....	71
5.5.4.1 AKILL	71
5.5.4.2 Weitere Opserv-Befehle	71
5.5.5 Botserv	72

5.5.6 Adminserv	72
5.6 Datenbank.....	72
5.6.1 Erstellen Datenbank.....	72
5.6.2 Laden der gespeicherten Daten.....	72
5.6.3 Speichern der Daten	73
5.7 Hilfe.....	74
6 Testing	75
6.1 Konzept	75
6.2 Test-Protokoll.....	75
6.3. Testing der Applikation	75
7 Ausblick.....	77
8 Fazit	78
9 Anhang.....	79
9.1 Anhang A: Bilderverzeichnis	79
9.2 Anhang B: Tabellenverzeichnis	80
9.4. Literaturverzeichnis.....	81

1 Einleitung

1.1 Themenwahl

Wir leben im Zeitalter von Web 2.0 wodurch der klassische Text-Chat an Bedeutung verloren hat. Schliesslich sind Web-Browser heute fähig Videos abzuspielen und Webcam Übertragungen durchzuführen. IRC als reiner Text-Chat spielt also nur noch eine untergeordnete Rolle bei den Anwendern.

Dennoch ist der IRC (Internet Relay Chat) in UNIX-Kreisen immer noch äusserst populär. Nehmen wir zum Beispiel den Chat auf irc.freenode.org. Dort finden sich auch heute noch zehntausende Benutzer die gleichzeitig online sind. Sie treffen sich themenbasierten Räumen wie zum Beispiel `#linux` oder `#C` und bilden so eine riesige Gemeinschaft um bei Fragen zu den jeweiligen Themen zu helfen.

```

[01:24:27] * Now talking in #c!
[01:24:27] * Topic is "The C Programming Language || C11 is the current C Standard || PASTE (>3 lines) here: http://ideone.com/ || HOME page: http://www.iso-9899.info/ (to edit you need an account, see a wiki/channel admin) || BOOKS here: http://www.iso-9899.info/wiki/Books || Be civil"
[01:24:27] * Set by pragma- on Mon Dec 22 18:58:57 2014
[01:24:27] * ##c Fish-Guts mikedugan r3g3x [spoiler] robot-beethoven doppel dm7freak Muzer neurodrone cousteau netj desantis arescorpio entitz schleppel Jubb Love4Boobies phinx
Alburi asakura frostschutz bolt Uffalizer kyllerisse1010 dardevlin mgorbach Trondby matp jeffieismyhero nistyre_ moop aslant immbis Saporok_ anonnumberanon Lowl3v3l plitter
zymurgy Pessimist oleo Tordek roentgen likecolacola BlueShark_ KindOne Enthralled vishwin n1000_ neuron moparstbest mulvane
[01:24:27] * ##c Mr_Sheesh Beethny wrl Photon12 KWhat4 DarthTetris Wamanuz Anaphaxeton raj TMM screwjack GinoMan delta-nry tigger0jk x1 dts[pokeball Jesin fengshaun Ephexev
FreezingCold rrm_ pnbearst shikhn MrCoffee cathaur NeoGeo64 flaviu junaidnaseer2 bxomonjhome tharugrim killertester Celelbi freeruncan Noldorin hat] raboof cpt_yossarian
hiptoeubic kesselhaus bone Khisanth BreadProduct Nothing4You Atrumx Haswell Biohazard BoredHamster Zentdayn heurist Suchorski Blaguvest
[01:24:27] * ##c Orion] sparetire_ neionz sjohnson tapout brenns10 jamiec fstd prc1 bluss DarkFazy M-ou-se krokus apeiros_ d3m1g0d-Left_Turn Synchunk fizzie neo1691 garner jzk
sharky ivan] day cebor_ Cabanossi Riviera Enissay greenbagels megaTherion araujo ChoiKySang turnedtodust whaletchno sivoais NIN101 nicedice ltd yarker Vutral Exagone313 Auv
Irrelum CHC heftig_ dexter0 KrzaQ Farow Jaood Norrin cjwelborn msy cysm Gravitrion slxpk rcombs jz] george2 Polarina bluehavana
[01:24:27] * ##c Bigcheese synapt notion Nidkeeh deste a-a skarn gbarboza ConstantineXVI viruser Atlantic777 CrypticSquared jgeboski drdanick cats toxboi_ pippijn xace kst notpara
mathewji terabit flounders sheilong Anoniem4 ss mnb_ CJKay BSaboia_ liberal l0rdn1x quinman22 kalzz bb010g Karethoth Fenhl doudcell hellschreiber superjudge caen23 ekneuss
starship Zerock nuopogdi Talsman twoface88 stuxjRC-only dorp jds camerin tsuyoi homebrew rud93 bpmedley R0b0t1 antoxyz lens
[01:24:27] * ##c Ard1t cpt-oblivious thomas CMGauger tempus_ fol kermi Nik05 lyindArk robbyconnor dusted Vohveli Innominate glebofff Natch ncjb widmo pHcF mervake keel vicenteH
davor edr ElectricSheep DarthGandalf asmx marvimias DJZOO! SM0TVI orbitz ark_ de-vri-es cyberspace- SiegeX Asadh drakored lytchi scrapcode ormaaj Lokomotiv james41382 Trivium
polygraphed BGL roboman2444 dx boru ishkawa m0shbear barfod [Awaxx] Spark BombStrike glukt Zmsky jayme Conquer1Warrior MrC
[01:24:27] * ##c Praise_ether_ SirCnpwin dozn Alina-malina Jagged rol31 w41 Steffanx iron_houzi kirin_ yayachiken Ulrike_Rayne kurti wapifapi tazle @candide iveqy bool_ ironode0
proZaC Inode Burgundy jriese j_wright Niamik riotz sabotender KnownSyntax toothrot Saban sabalaba nitrix pestle benwaffle byaruhaf bdamos gf3 sono Brando753 freanux sbahra
stonecolddevin dagle myte: Defaultti izabera xpog tormoz KMA gojdfish zid_ Kostenko_ introom froggyman pragma- lanhedoesit
[01:24:27] * ##c kniskropd freakazoid0223 dami Caveddude Number 2 klarrt BSDBlack kate elyraz notker RichiH letoram shibumi diethyl fishes1 1 robink nukeddx blackbit alip ashan kbt
DarkCthulhu kiddo Tabmow jrib Trieste Thermi Taylor genpaku BlastHardcheese Trashlord paracyst rob_ _ refx_ cherwin spoty staykov renopt hohum jv Plazma Platini talanor dh
Robdgreat Aero Vorpal croz XgF phantomcircuit huntercool relaxed Xgc RTG_ Triskelos lykkn mthowie altf2o tm512 pumba MindlessDrone
[01:24:27] * ##c Mellett68 Shayanjm yokel tomaw tajnyman ccaione cross rascal999 catern fold Chris tsstc andres Kasreyn dostoyevsky AndrewX192 geidi rvncerr impulse150 marky LoRez
Happzz tobiasBora Puculowski debris_ thomney monokrome ra4king dualbus HylianSavior haasn fatpudding pandzila LiamM gl barometz Fill kline Buzzer tomodachi rigid goncalo mofino
Greatgib icedev Hearbroken sparetire CaZe baggio Cerise flam_ kloeri heinrich5991 caveat- dav1d impulse emias mountaingoat
[01:24:27] * ##c Nagra shvouchk awake iptable Bane_ hennilu DarkGhost arch_ dutc koolman jiffe beneth_ keltvek chewbranca leprechau rolt SporkWitch bitdigger pa Cleo kov
salamanderrake chishiki csuiu boos intoX27h pstef ski bumber_ Yaknot5 f0lder LeoNerd ohama richardwhiuk oblique Cyclohexane octanium urlgrabber Virox K1773R S_T_A_N jon_y
epsylon hrnz monty metalys SHODAN octe b00k3r Affliction Gentle codebrainz Paradox924k bps Zarthus ribasushi Evil3Stoker Astorm Tutkah
[01:24:27] * ##c dtsf anunnaki Draconx Stary2001 shiftplusone mischief machty_ Jupelius kotthog dxtx mit averell KingOfKarlsruhe edk aandrew babilen wio7 gagan662 APic unreal Zhivago
Thanat0s_cnu- MatheusOI sharpnelli dv- wdouglas nutron Fuux yorick kern edwardly twkm seg ivan_ stfn kAworu sazzter timemage divine Ewomire FergusL elwinto Oddity petern_
[01:24:27] * Users: [ @Ops:1 +Voiced:0 Regular:555 Total:556 ]
[01:24:27] * Modes: +Cnrtf #c-unregistered
[01:24:27] * 1164523359
[01:24:28] * http://www.iso-9899.info/
[01:24:31] <doppel> which means i'll be around to bug you for years to come. how lucky are you.

```

Abb. 1: Chatraum auf irc.freenode.org

Auch wenn die Benutzer heute auf andere Chat-Technologien ausweichen heisst das nicht, dass der IRC am Aussterben ist. Viele Benutzer in der Linux-Welt machen sich nichts aus grafisch hochstehenden Flash-Chats sondern möchten schnell und einfach Informationen austauschen.

Da jedoch nicht immer alle Benutzer freundliche Absichten haben und sich teilweise daneben benehmen benötigt man Mittel um sich solch ungebeter Gäste zu entledigen. Dazu dienen sogenannte Operatoren in Räumen. Sie haben Rechte, die „normale“ Benutzer nicht haben. So können Operatoren zum Beispiel einen Benutzer „kicken“, also aus dem Raum werfen oder „bannen“ also verbannen, sodass dieser Benutzer den Raum nicht mehr betreten kann.

Das Protokoll auf dem IRC zugrunde liegt sieht vor, dass der erste Benutzer der den Raum betritt automatisch den Operatoren-Status erhält. Er kann weitere Operatoren bestimmen und hat weitere Rechte.

Dadurch entsteht jedoch das Problem, dass wenn nur ein Operator im Raum ist und er diesen verlässt niemand mehr Operatoren-Rechte hat, bis alle weiteren Benutzer den Raum verlassen und dieser so neu erstellt werden kann.

Ein weiteres Problem ist, dass das Protokoll keine Speicherung von Benutzernamen (sog. „Nickname“) zulässt. Wenn also ein Benutzer den gewünschten Namen bereits innehat gibt es keine Möglichkeit denselben Namen zu verwenden.

Um diesen und weiteren Problemen zu begegnen wurden sogenannte „Services“ erschaffen. Diese bieten zum Beispiel die Möglichkeit, Nicknames oder Chaträume (sog. Channels) zu registrieren und so vor unbefugtem Zugriff zu schützen.

In einigen bestehenden Lösungen sind diese bereits enthalten, jedoch gibt es zahlreiche Server, die ohne Services ausgeliefert werden.

Der Autor der vorliegenden Arbeit war lange Zeit als Moderator auf dem Chat von Swisscom (damals Bluewin) tätig. Dieser Chat lief auf einer proprietären Lösung mit integrierten Services. Nach der Abschaltung des Chat begannen einige Nutzer eigene Chats aufzusetzen, dies mit Open-Source Lösungen.

Da viele Open-Source Lösungen keine integrierten Services haben, stellte sich so bald die Frage nach einer entsprechenden Lösung. Es gibt einige gute und populäre Services-Lösungen, jedoch unterscheiden diese sich stark und viele Benutzer wünschten sich, dass die Befehle ähnlich wie auf dem Bluewin Chat aufgebaut seien. Um dieses Bedürfnis zu befriedigen entschloss sich der Autor eine entsprechende Lösung zu entwickeln. Die vorliegende Arbeit befasst sich mit dieser Lösung.

1.2 Ziele der Arbeit

Um die im Kapitel 1.1 beschriebenen Probleme zu lösen und die Bedürfnisse zu befrieden soll ein Services-Server geschaffen werden, der sich zu einem bestehenden IRC Chat verbindet und diese Dienste anbietet. Die Umsetzung soll in der Programmiersprache C erfolgen. Namentlich soll folgende Dienste angeboten werden:

- **Nickserv:** Dienst für die Registrierung eines Nickname um diesen mittels Passwort vor der Verwendung durch andere zu schützen.
- **Chanserv:** Dienst für die Registrierung eines Channels. Zudem wird die Möglichkeit geschaffen, Operatoren zu registrieren, die sich in einem registrierten Channel von Chanserv die Operatoren Berechtigung selber geben können.
- **Botserv:** Dienst für die Erstellung von Bots die einen Channel offenhalten sollen.
- **Opserv:** Dienst für Benutzer mit erweiterten Rechten (IRC Operatoren, Administratoren).
- **Adminserv:** Dienst für Administratoren um bestimmte Operationen wie z.B. das Speichern der Datenbank vor einem Neustart durchzuführen.

Um keine Abhängigkeit von einem Datenbankserver zu erzeugen sollen die Daten in einer SQLite-Datenbank gespeichert werden. SQLite eine Datenbank die Dateibasiert ist und SQL Syntax unterstützt. Der gesamte Datenbestand soll beim Starten des Servers in den Arbeitsspeicher geladen und in regelmäßigen Abständen gespeichert werden.

Die spezifischen Einstellungen des Servers und der Dienste sollen konfigurierbar sein. Des Weiteren soll für jeden Befehl der gesendet werden kann eine Hilfedatei erstellt werden die den jeweiligen Befehl erklärt.

1.3 Aufgabenstellung

Um die in Kapitel 1.2 genannten Ziele zu erreichen werden folgende Teilschritte durchgeführt:

- Marktanalyse: Feststellung bereits existierender Lösungen
- Studium des IRC Protokolls (RFC 1459) um die protokollarischen Anforderungen zu erfassen.
- Aufnehmen der Anforderungen: Welche Befehle soll ein Dienst unterstützen und wie sollen diese funktionieren?
- Festlegung der Grenzen: wie viele Einträge sollen Listen maximal haben dürfen?
- Erarbeiten der Grundlagen für Timer und Signale welche für Timeouts benötigt werden.
- Durchführung des Requirement Engineering
- Ausfertigung des technischen Entwurfs
- Implementierung Basis-Server
- Implementierung der Dienste
- Testing inkl. Ausarbeitung eines Testkonzepts und Testprotokolls
- Dokumentation

2 Einführung in IRC und Services

2.1 Geschichte des IRC

Das IRC (Internet Relay Chat) stammt noch aus den Anfängen des heutigen Internets. Um 1988 wurde das ARPAnet außer Betrieb genommen und das NSFnet übernahm die Funktionen. Hierdurch wurde das TCP/IP Protokoll erschaffen und auseinander liegende Institutionen konnten miteinander kommunizieren.¹

Die ursprüngliche Idee eines Chat-Netzwerkes entstand im BITNET unter dem Namen Relay Chat. Dieses System wurde vom finnischen Studenten Jarkko Oikarinen, der an der Fakultät für Informatik der Universität Oulu studierte, im Sommer 1988 auf das Internet übertragen.²

Durch einen Freund Oikarinens, erfuhren Universitäten in den Vereinigten Staaten von dem Chat. Es folgte eine Anfrage, ihren Server mit dem finnischen zu verbinden. So verliess der IRC zum ersten Mal Finnland.

Bereits 1989 gingen die ersten deutschen Universitäten mit Chats ans Netz und einige geschichtliche Ereignisse trugen in der Vergangenheit in besonderem Maße zur Popularität des IRC bei. Während des Golfkrieges im Jahr 1991 waren die aktuellsten Berichte über den Verlauf des Krieges über den IRC erhältlich. Ein ähnliches Szenarium spielte sich im September 1993 ab, als gegen das sowjetische Staatsoberhaupt, Boris Yeltsin, geputscht wurde. IRC-Benutzer aus Moskau erzählten live von den dortigen Ereignissen.³

Im Jahr 2000 ging in der Schweiz der Bluewin Chat online. Zu ihren besten Zeiten waren über 6'000 Benutzer gleichzeitig online und bildeten so den populärsten Chat der damaligen Zeit.

Seit Mitte der 2000er Jahre die Flash basierten Chats an Popularität gewannen gingen die Nutzerzahlen auf dem IRC Netzwerken zurück. Die grossen Netzwerke wie zum Beispiel QuakeNet oder FreeNode erfreuen sich jedoch immer noch grosser Beliebtheit.

2.2 Funktionsprinzip des IRC

Mehrere IRC Server können sich zu einem Netzwerk zusammenschliessen. So können die Benutzer die Benutzer von allen Servern miteinander agieren.

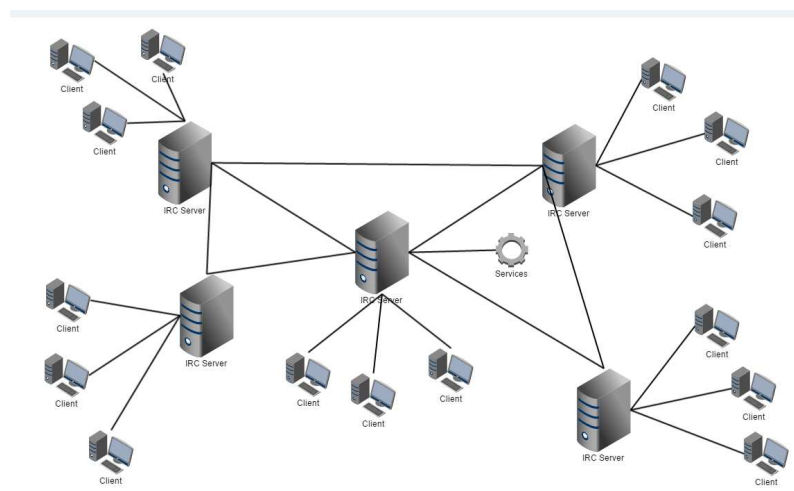


Abb. 2: Schema eines IRC Netzwerkes

Wie in Abb. 2 zu sehen ist, verbinden sich die Clients jeweils zu einem Server. Die Server schliessen sich zu einem Netzwerk zusammen. Jeder Benutzer kann so alle anderen Benutzer erreichen. Das Gleiche gilt für die Services. Die Services sind im Prinzip nichts anderes als ein weiterer Server, der sich mit einem der im sich Netzwerk befindlichen Servern verbindet.

Damit ein Benutzer chatten kann benötigt er einen Client. Diese sind in der Regel kostenlos erhältlich. Populäre Clients sind z.B. mIRC für Windows oder XChat für Linux. Das eigentliche Chatten geschieht über das Senden von Befehlen. Möchte ein Benutzer eine Nachricht an einen anderen Benutzer schicken, so tippt er z.B. /msg Benutzer Nachricht

Der Slash (/) signalisiert dem Client so, dass ein Befehl abgesetzt werden soll. Er wandelt den erhaltenen Befehl dann in das Format um, das vom Protokoll verlangt wird. Die genaue Syntax für die Eingabe kann von Client zu Client variieren, die endgültige Form wird jedoch vom Protokoll definiert.

2.3 Protokoll

Um eine einheitliche Form der Nachrichtenübermittlung zu gewährleisten sind die IRC Befehle in einem Protokoll genau definiert.

Somit wird sichergestellt, dass die Nachrichtenübermittlung mit jedem Client und jedem Server in einer strukturierten Art und Weise zuverlässig funktioniert.

Das Protokoll für IRC ist als RFC 1459 (Request for Comments) definiert. Mit einem RFC gibt man ein Dokument für eine Diskussion frei. Es behält den Namen RFC auch dann, wenn es allgemeine Akzeptanz erreicht hat.

Nachfolgend wollen wir einige wichtige Punkte des Protokolls betrachten.

2.3.1 Einführung

2.3.1.1 Grundlagen

Das Protokoll legt fest, dass jegliche Nachrichten die über das Netzwerk gesendet werden, aus den drei Teilen

- Präfix (optional)
- Befehl
- Parameter

bestehen und folgendes Format aufweisen müssen:

```
:Präfix Befehl Argument CRLF
```

Zwischen dem Kolon (:) und dem Präfix darf kein Leerzeichen stehen und jeder Befehl muss mit Carriage Return (CR) und Line Feed (LF) abgeschlossen werden.

Der Befehl muss ein gültiger IRC Befehl oder dessen numerische Repräsentation sein. Eine Liste gültiger Befehle kann auf <https://tools.ietf.org/html/rfc1459>.

Wir dieses Format nicht eingehalten, wird der Befehl vom Server ignoriert. Auf die meisten Befehle soll der Server eine Antwort senden. Diese hat eine eindeutige Nummer und ist im Kapitel 6 des Protokolls definiert.

2.3.1.2 Operatoren

Um eine grundlegende Ordnung aufrecht zu erhalten kann der Betreiber des IRC Servers sogenannte Operatoren ernennen. Die haben im Vergleich zu anderen Benutzern erweiterte Rechte. Diese sind nicht mit Operatoren in Chaträumen zu verwechseln, welche Störenfriede aus dem Chatraum verbannen können. Vielmehr haben diese Operatoren Rechte die für den ganzen Server gelten. So können diese zum Beispiel einen Benutzer von gesamten Netzwerk verbannen.

2.3.1.3 Channels

Ein Channel ist ein Chatraum. Also ein Gebilde in dem sich mehrere Benutzer treffen und austauschen können. Nachrichten die an einen Channel gesendet werden können von allen Benutzern in diesem Channel gelesen werden. Einen Channel kann man mit dem Befehl JOIN betreten.

Wie in Kapitel 2.3.1.2 erwähnt gibt es neben den Server-Operatoren auch Channel-Operatoren die unerwünschte Benutzer aus dem Channel verbannen können. Diese Operatoren haben auch weitere Rechte, zum Beispiel können diese auch weitere Benutzer zu Operatoren ernennen.

2.3.2 Die IRC Spezifikation

Für IRC wurde kein spezifischer Zeichensatz definiert. Vorgegeben ist lediglich, dass ein 8-Bit Zeichensatz verwendet muss. Jedoch hat sich das amerikanische ASCII als Standard durchgesetzt. Das heisst, dass in der Regel keine Umlaute für Server Befehle verwendet können. Dasselbe gilt für Benutzernamen, sogenannte Nicknames.

Die Kommunikation zwischen den Benutzer wird hingegen nicht beeinträchtigt. Dort können jedwede Art von Zeichen gesendet werden, da die meisten Server für die Kommunikation UTF-8 unterstützen.

2.3.3 IRC Konzepte

In diesem Kapitel untersuchen wir die protokollarischen Konzepte der Nachrichtenübermittlung basierend auf dem nachfolgenden Bild.

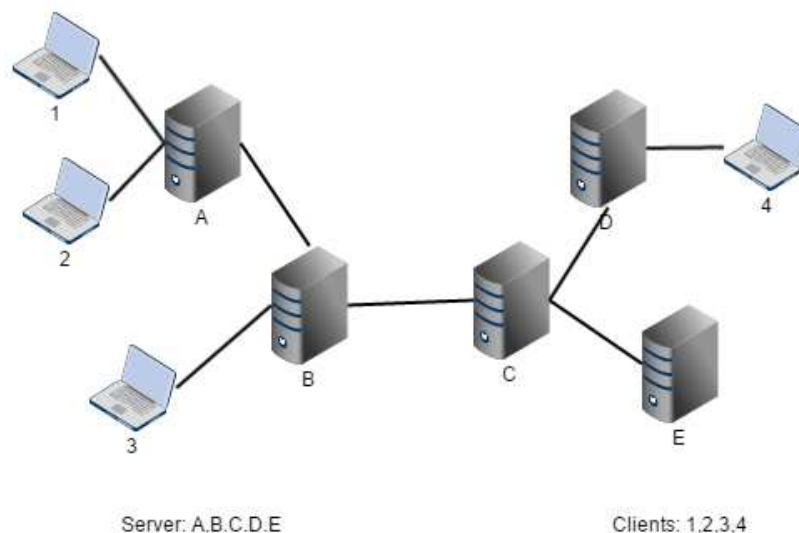


Abb. 3: Struktur eines IRC Netzwerks

2.3.3.1 1:1 Kommunikation

Die Kommunikation auf einer 1:1 Basis erfolgt in der Regel zwischen Benutzern, da die Server nicht ausschliesslich untereinander kommunizieren.

Der Pfad einer Nachricht nimmt stets den kürzesten zwischen zwei Punkten im Netzwerk. 1:1 Nachrichten werden nur von Sender, Empfänger und den Servern, die die Nachricht weiterleiten sollen gesehen.

Beispiel 1: Eine Nachricht zwischen Client 1 und 2 wird nur vom Server A gesehen, der diese direkt an Client 2 schickt.

Beispiel 2: Eine Nachricht zwischen Client 1 und 3 wird von den Server A und B gesehen.

Beispiel 3: Eine Nachricht zwischen Client 2 und 4 wird von den Servern A, B, C und D gesehen.

2.3.3.2 1:n Kommunikation

Die 1:n Kommunikation auf dem IRC tritt in verschiedenen Formen auf. Diese sind:

- Nachricht an eine Liste: Der Client schickt die Nachricht an eine Liste von Empfängern.
- Nachricht an eine Gruppe: Der Client schickt eine Nachricht einen Channel
- Nachricht an einen Host: Der Client schickt eine Nachricht Empfänger welche über die gleiche Hostmaske verfügen. Beispiel: Eine Nachricht an `*@*.swisscom.com` empfangen alle, die über einen Swisscom-Anschluss auf dem Server verbunden sind. Diese Art der Nachricht ist üblicherweise Operatoren vorbehalten.

2.3.3.3 1:Alle Kommunikation

Diese Art der Kommunikation beinhaltet Nachrichten die an alle Clients oder Server oder beides verschickt werden. Dies kann ein hohes Datenvolumen verursachen.

2.3.3 Nachrichten Details

Damit sich ein Client zu einer Server verbinden kann, muss sich der Client nach Erstellung der Verbindung mit dem Server registrieren. Dies geschieht über die Übermittlung von Verbindungsnachrichten. Die nachfolgenden Nachrichten müssen demnach gesendet werden.

2.3.3.1 Pass Nachricht

Diese Nachricht ist nur dann erforderlich, wenn der Server mit einem Passwort geschützt ist. Folgende Syntax ist zwingend:

```
PASS <Passwort> CRLF
```

2.3.3.2 Nick Nachricht

Mit diesem Befehl legt man seinen eigenen Nicknamen fest.

```
NICK <Nickname> CRLF
```

Wird der Nickname bereits verwendet meldet der Server die Fehlermeldung

```
ERR_NICKNAMEINUSE
```

zurück.

2.3.3.3 User Nachricht

Mit dieser Nachricht werden die Details des Client an den Server gemeldet.

```
USER <Benutzername> <Hostname> <Servername> <Realname>
```

Der Benutzername ist Teil der Hostmaske. Diese wird in den Einstellungen im Programm zum Chatten in der Regel als E-Mail Adresse angegeben und dem Server so übermittelt.

Der Hostname wird durch den Internetanbieter bestimmt. Er ist von der IP Adresse abstrahiert.

Der Servername wird automatisch gesetzt. Verbindet man sich auf ein Netzwerk mit mehreren Servern ist hier der Name des Servers über den man sich verbindet gemeint.

Der Realname kann frei festgelegt werden. Er kann öffentlich eingesehen werden, daher sollte man sich gut überlegen, ob man seinen echten Namen angeben will.

Als Beispiel können wir den Windows Client mIRC hinzuziehen. Bevor man sich auf einen Chat verbindet muss man in den Einstellungen diese Informationen hinterlegen:

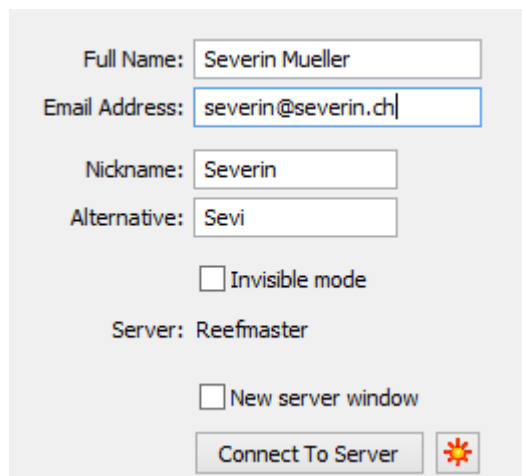


Abb. 4: Verbindungseinstellungen mIRC

Der Full Name ist der Realname und die E-Mail Adresse liefert den Benutzername. Jedoch hat nur der Teil vor dem @ einen Einfluss.

Wie erwähnt können wir den Hostnamen nicht beeinflussen. Wenn uns diese Information interessieren sollte können wir nach dem Verbinden eine WHOIS Nachricht an den Server schicken. Mit WHOIS werden diverse Information über den angegebenen Benutzer angezeigt. Syntax

```
WHOIS <Nickname>
```

Wenn wir also im Beispiel Severin den Hostnamen herausfinden möchten, senden wir die Nachricht /whois Severin

an den Server. Folgendes kommt zurück:


```
[16:57:31] * Severin is connecting from *@185-103.60-188.cust.bluewin.ch 188.60.103.185
[16:57:31] * /whois Info Severin
[16:57:31] * Severin is ~severin@185-103.60-188.cust.bluewin.ch
[16:57:31] * Realname: Severin Mueller
[16:57:31] * Severin on #c-unregistered
[16:57:31] * Severin using sendak.freenode.net, Vilnius, Lithuania, EU
[16:57:31] * Severin has been idle 11secs, signed on 6secs ago
[16:57:31] * Severin End of /WHOIS Info.
[16:57:31] -
```

Abb. 5: Rückgabe einer WHOIS Nachricht

Der Hostname ist hier gelb markiert.

2.3.3.4 Server Nachricht

Wollen wir statt einen Client einen Server verbinden, so müssen wir eine Server Nachricht schicken.

```
SERVER <Servername> <Hopcount> <Info> CRLF
```

Servername ist der Name des Servers den wir mit dem Netzwerk verbinden wollen.

Hopcount ist eine interne Information die indiziert, wie weit die Server voneinander weg sind (wie viele Knoten dazwischen liegen)

Info ist eine Beschreibung des Servers.

2.3.3.5 PRIVMSG

Um eine Nachricht an einen Client oder Server zu senden, verwenden wir PRIVMSG.

```
PRIVMSG Empfänger :Nachricht CRLF
```

Auch Dienste, die in diesem Projekt implementiert werden, werden mit dieser Nachricht angesteuert.

2.3.3.6 NOTICE

Eine weitere Möglichkeit eine private Nachricht zu senden ist NOTICE. Üblicherweise wird in diesem Fall kein neues Chat-Fenster geöffnet, sondern die Nachricht wird im aktiven Fenster angezeigt.

```
NOTICE Empfänger :Nachricht CRLF
```

Dienste wie in diesem Projekt senden ihre Antworten üblicherweise als NOTICE.

Auf weitere Details des Protokolls verzichten wir an dieser Stelle. In den späteren Kapiteln dieser Arbeit werden wir auf die benötigten Teile des Protokolls eingehen.

2.4 Services

Wie in Kapitel 1 erwähnt bietet das IRC Protokoll an sich keinerlei Funktionen um einen Nicknamen oder Channel zu reservieren. Für diesen Zweck wurden Services geschaffen.

Nachfolgend möchten wir erläutern, was genau wir unter diesen Services verstehen.

2.4.1 Einführung

Services sind im Grunde nichts anderes als ein Server der sich zum IRC Netzwerk verbindet. Auf diesem Server laufen die einzelnen Dienste wie normale Clients, mit dem Unterschied, dass diese eine Kennung besitzen, die sie als Service kennzeichnet. Diese Kennung wird als sogenannter Usermode vom Server zur Verfügung gestellt (siehe Kapitel 4.2.3.2 im IRC Protokoll).

Es gibt keine allgemeingültige Definition welche Services ein Server unterstützen soll. Das Protokoll legt lediglich fest, dass ein Benutzer als Service markiert werden kann.

In der Praxis haben sich jedoch einige Arten von Services durchgesetzt. Dazu zählen:

- Nickserv: Dienst um Nicknames zu registrieren
- Chanserv: Dienst um Channels zu registrieren
- Memoserv: Dienst um Nachrichten an registrierte Benutzer zu senden
- Opserv: Dienst für Server-Operatoren
- Botserv: Dienst für das Erstellen von Bot die als eine Roboter fungieren
- Hostserv: Dienst um für bestimmte Benutzer neue Hosts zu setzen
- Adminserv: Dienst für Server Administratoren

In dieser Arbeit befassen wir uns nur mit Nickserv, Chanserv, Opserv, Botserv und Adminserv.

2.4.2 Nickserv

Wie erwähnt stellt Nickserv einen Dienst zur Verfügung mit dem man sich einen gewünschten Benutzernamen registrieren kann. Üblicherweise wird dieser mit einem persönlichen Passwort geschützt. Da heisst, dass man sich für diesen Nick identifizieren muss, bevor man ihn verwenden kann. In der Regel werden weitere Funktionen zur Verfügung gestellt. Meist sind dies Optionen die mit anderen Diensten interagieren.

2.4.3 Chanserv

Chanserv oder Channel Services sind ein Dienst für die Registrierung von Channels. Mit einer Registrierung soll sichergestellt werden, dass unbefugte Benutzer den Channel nicht „kapern“ können. Unter kapern verstehen wir das erschleichen von Op-Rechten um dann den Channel zu übernehmen. Da das IRC Protokoll vorsieht, dass jeder Benutzer einem anderen Benutzer Op-Rechte geben kann kam dies häufig vor. Mittels einer Registrierung kann sich der Eigentümer des Channels jederzeit über Chanserv selber die Op-Rechte zurückgeben lassen und so sicherstellen, dass der Channel nicht gekapert wird.

2.4.4 Opserv

Opserv soll Server Operatoren die Möglichkeit geben, einige Operation global, also auf allen Server des Netzwerkes zu steuern. Wir wollen einen Schritt weitergehen und Administratoren die Möglichkeit geben, normalen Benutzern ohne Operatoren Recht Zugriff auf Opserv zu erlauben. Dies kann z.B. nützlich sein, wenn kein Operator anwesend ist und man einen Benutzer dem man vertraut die Rechte geben kann damit dieser vorläufig nach dem Rechten sieht.

2.4.5 Botserv

Es kann vorkommen, dass auf gewissen Netzwerk Channels die nicht registriert wurden eine Möglichkeit geben möchte, ein Takeover (kapern) des Channels zu verhindern, ohne dass der Channel registriert werden muss. Mit Botserv kann man sogenannte Bots erstellen, die die Aufgabe haben, einen Channel offenzuhalten und ein Takeover zu verhindern. Ein Bot wird mit einem Passwort versehen, mit welchen man sich für den Bot identifizieren kann und so auf die Funktionen Zugriff erhält.

2.4.6. Adminserv

Mit Adminserv kann man Administratoren erlauben, gewisse Server Operationen durchzuführen. Diese können z.B. ein Neustart der Services, das Speichern der Datenbanken oder das Abschalten der Dienste sein.

Die genaue Implementierung kann sich zwischen den einzelnen Lösungen stark unterscheiden. Wir möchten daher untersuchen, welche Lösungen bereits existieren wo es Lücken gibt, die es zu schließen gilt.

2.4.7 Festlegung der Grenzen

Um eine gewisse Stabilität und Performance zu gewährleisten müssen wir einige Grenzen festlegen. Folgende Limitationen sind zu berücksichtigen:

- Die Befehlslänge wird auf 1024 Zeichen begrenzt
- Die Länge von Nickname und Channels wird durch das Protokoll auf 32 Zeichen begrenzt
- Das Projekt ist zunächst nur für Linux zu implementieren

3 Marktanalyse

Wie in der Computerwelt üblich gibt es auch für IRC Services und Server verschiedene Lösungen. Einige davon sind proprietäre, andere Open-Source Lösungen. Wir möchten in diesem Kapitel einige davon genauer betrachten und feststellen, wo unsere Lösung bestimmte Lücken füllen kann.

3.1 Webmaster ConferenceRoom

ConferenceRoom von Webmaster ist eine proprietäre Lösung die eine All-in-One Lösung bietet. Das heisst, dass der IRC Server und die Services in einem Produkt enthalten sind.

Der Vorteil dieser Lösung liegt in den integrierten Services und der Stabilität der Software.

Der Nachteil sind die eher hohen Kosten. Eine Version für 1000 Benutzer kostet \$250.00 und die Version für 10000 Benutzer kostet \$995.00⁴

Die 1000 Benutzer Version bietet zudem nur die Dienste Nickserv und Chanserv an. Möchte man Memoserv dazu haben benötigt man die teurere Version.

Der Chat von Bluewin wurde auf einem Conference Room Server betrieben.

3.2 Unreal IRCd

Unreal IRCd ist ein Open Source IRC Server der zu den populärsten Servern überhaupt gehört. Er läuft äusserst stabil, bietet eine Vielzahl an Konfigurationsmöglichkeiten und ist aussergewöhnlich gut dokumentiert. Als Open-Source Lösung ist der Server kostenlos erhältlich. Wenn man den Server jedoch modifiziert darf man keine technische Unterstützung erwartet.

Ein Nachteil dieser Lösung ist, dass keine Services mitgeliefert werden. Diesem Problem kann man begegnen indem man auf eine externe Lösung zurückgreift. Nachfolgend betrachten wir einige Open-Source Services.

3.3 IRC Services

Die IRC Services sind quasi das Original. Praktisch alle populären Lösungen basieren auf dem original Source Code von Andrew Church. Die erste Version wurde 1996 entwickelt. Diese Service werden nicht mehr weiterentwickelt. Die allererste Version dieser Lösung ist immer noch online auf <http://achurch.org/services/> zu finden.

Da die Entwicklung vor einiger Zeit eingestellt wurde bieten diese Services nicht den Umfang die moderne Systeme bieten.

3.4. Anope Services

Anope ist eine Services Lösung die auf den originalen IRC Services basiert. Anope bietet eine Vielzahl an Optionen und läuft äusserst stabil. Anope verfügt über eine gute Dokumentation und 6 Services. Zudem werden Module, die einfach dazu geschaltet werden können, unterstützt.

Der Funktionsumfang von Anope ist beträchtlich.

Ein Nachteil ist jedoch, dass sich die Syntax und die Ausgaben teilweise stark von denen die ConferenceRoom bieten unterscheiden und ehemalige Swisscom-Chatter Mühe haben, sich zurecht zu finden.

3.5 Auspice

Auspice sind eine Open-Source Lösung die auch unter Windows funktioniert. Leider wurde die Entwicklung 2005 eingestellt. Sie sind aber nach wie vor zum Download erhältlich und auch in Betrieb.

3.6 Fazit

An den heute erhältlichen IRC Services ist zu sehen, dass IRC an Popularität eingebüsst hat. ConferenceRoom und Anope werden weiterentwickelt, genau wie auch Unreal IRCd, die weiteren Services werden nicht mehr wirklich gepflegt.

Da viele der ehemaligen Swisscom Chatter sich wünschen, dass die Services ähnlich aufgebaut sind wie diejenigen von ConferenceRoom soll unsere Lösung dieses Ziel verfolgen.

4 Anforderungen

In diesem Kapitel wollen wir die Anforderungen für die Services erfassen. Zunächst wollen wir die Use Cases beschreiben. Aufgrund der Use Cases erstellen wir die Anforderungen. Als Akteure sollen die Stakeholder dienen die wir wie folgt definieren.

4.1 Basis-Server

Wir definieren die Akteure für den Basis-Server wie folgt

Akteur	Beschreibung
Server Administrator	Kann den Server konfigurieren und verfügt über alle Rechte
Basis-Server	Der Basis auf dem die Services laufen. Verbindet sich zum IRC Server

Tabelle 1: Stakeholder

Als Basis Server wird der eigentlich Server bezeichnet, auf dem die Services laufen sollen. Der Server soll konfigurierbar sein, demnach muss das Programm einen Mechanismus zur Konfiguration bereitstellen.

4.1.1 Use Case Basis Server (UC S-01)

Use Case Basis-Server

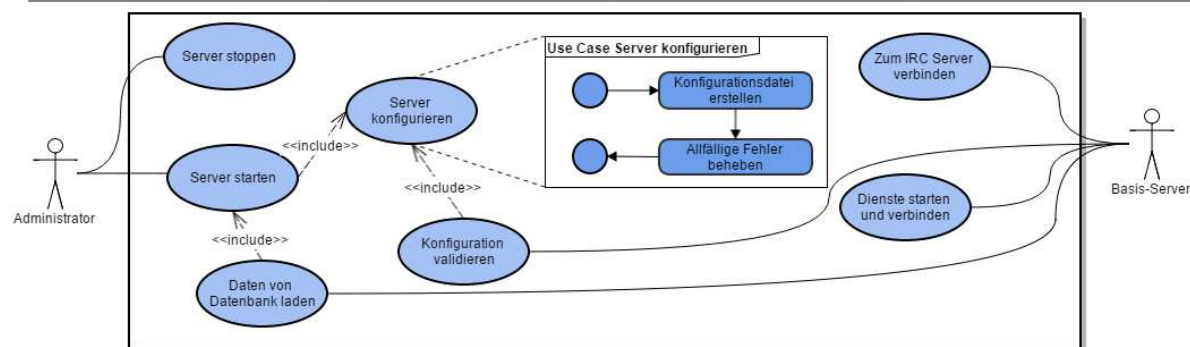


Abb. 6: Use Case Basis Server

Die Benutzer schicken ihre Nachrichten an den IRC Server und nicht an den Services-Server. Der IRC Server leitet dann die Nachricht an den richtigen Service weiter.

4.1.2 Anforderungen Basis-Server

4.1.2.1 IRC Server

Zunächst müssen wir festlegen, welche Chat-Server überhaupt unterstützt werden sollen. Die Entscheidung fiel auf den Unreal IRCd Server, da diese Lösung die populärste ist und über eine gute dokumentierte Unterstützung für Services-Schnittstellen verfügt. Zu beachten ist dabei, dass das Unreal IRCd wie alle Server eine eigene Protokoll-Implementierung nutzen. Für sämtliche Nachrichten an den Server ist dies zu beachten. Das IRCd-Protokoll im Projektordner „unreal“ zu finden.

Daraus resultiert folgende Anforderung:

Requirement Nr.	R-S-001
Titel	Unterstützte IRC Server
Beschreibung	Der Unreal IRCd Server soll unsere Services unterstützen. Es sind die Grundlagen zu schaffen, dass der Basis-Server sich zu einem bestehenden Unreal IRCd Server verbinden kann.
Begründung	Unreal IRCd ist der populärste Open-Source IRC Server. So werden die Services einem breiten Publikum angeboten
Kommentare	

Tabelle 2: R-S-001: Unterstützte IRC Server

4.1.2.2 Konfiguration Basis-Server

Die Services sollen einen hohen Grad an Flexibilität aufweisen. Daher ist es erforderlich, dass alle wichtigen Werte des Basis-Servers konfiguriert werden können.

Requirement Nr.	R-S-002
Titel	Konfiguration Basis-Server
Beschreibung	Die Grundeinstellungen des Basis-Servers sollen konfigurierbar sein. Namentlich sind dies: Name des Basis-Servers Beschreibung Username und Hostname Post Adresse des IRC Servers Name des IRC Servers Passwort für die Verbindung Die Konfiguration soll als Text-Datei vorliegen.
Kommentare	

Tabelle 3: R-S-002: Konfiguration Basis-Server

4.1.2.3 Starten und Stoppen des Servers

Das Starten und Stoppen des Servers soll über ein Argument auf der Kommandozeile erfolgen.

Requirement Nr.	R-S-003
Titel	Starten und Stoppen des Servers
Beschreibung	Die Services sollen von der Kommandozeile gestartet und gestoppt werden können. Als Argument für den Programmstart wird „start“ festgelegt Als Argument für das Stoppen wird „stop“ festgelegt.
Kommentare	

Tabelle 4: R-S-003: Starten des Servers

4.1.2.4 Validierung Server-Konfiguration

Da es wichtig ist, dass in der Konfiguration nur gültige Werte stehen soll das Programm beim Start einen Fehler ausgeben wenn die Konfiguration nicht ordnungsgemäss validiert werden kann.

Requirement Nr.	R-S-004
Titel	Validierung Server-Konfiguration
Beschreibung	Der Server soll die Konfiguration beim Programmstart validieren und einen Fehler im Falle ungültiger Werte ausgeben. Die Zeilennummer der Konfiguration und eine aussagekräftige Fehlermeldung sind auszugeben.

Kommentare	
------------	--

Tabelle 5: R-S-004: Validierung Server-Konfiguration

4.1.2.5 Daten von Datenbank laden

Aus Performancegründen sollen die Daten im Arbeitsspeicher gehalten werden. Daher müssen die Daten beim Programmstart von der Datenbank in den Arbeitsspeicher geladen werden.

Requirement Nr.	R-S-005
Titel	Validierung Server-Konfiguration
Beschreibung	Der Server soll sämtliche Daten beim Programmstart von der Datenbank mittels SQL Query aus der SQLite Datei lesen und in den Arbeitsspeicher laden.
Kommentare	

Tabelle 6: R-S-005: Daten von Datenbank laden

4.1.2.6 Verbindung zum IRC Server

Der Basis-Server soll automatisch die Verbindung zum IRC Server erstellen. Dies soll aufgrund der Konfiguration erfolgen, ohne dass der Administrator weitere Eingaben tätigen muss.

Requirement Nr.	R-S-006
Titel	Verbindung zum IRC Server erstellen
Beschreibung	Der Server soll automatisch die Verbindung zum IRC Server erstellen. Dafür muss zunächst ein Socket geöffnet und eine Verbindung erstellt werden. Danach müssen die vom Protokoll verlangten Nachrichten im korrekten Format aufbereitet und an den Socket gesendet werden. Sind alle Nachrichten korrekt übertragen worden soll die Verbindung bis zum Programmende stehen. Tritt ein Fehler auf, soll dieser entsprechend ausgegeben und das Programm beendet werden.
Kommentare	

Tabelle 7: R-S-006: Verbindung zum IRC Server

4.1.2.7 Dienste starten und verbinden

Requirement Nr.	R-S-007
Titel	Dienste starten und verbinden
Beschreibung	Da die Dienste im Prinzip spezielle Clients sind, sind diese wie normale Benutzer auf dem Server zu registrieren. Die entsprechende Nachricht muss aufbereitet und an den Server gesendet werden. Siehe dazu Kapitel 3.1 im IRC Protokoll
Kommentare	

Tabelle 8: R-S-007: Dienste starten und Verbinden

4.1.2.8 Daten in regelmässigen Abständen speichern

Sobald die Verbindung zum IRC Server steht müssen die einzelnen Dienste gestartet und verbunden werden.

Requirement Nr.	R-S-008
Titel	Daten in regelmässigen Abständen speichern
Beschreibung	Die Daten sollen im in der Konfiguration festgelegten Intervall in der Datenbank gespeichert werden.

	Dazu ist ein Mechanismus zu erschaffen, der alle zwei Sekunden ein Signal an den Server schickt, damit dieser jedwede Art von Intervallen überprüfen kann.
Kommentare	

Tabelle 9: R-S-008: Daten in regelmässigen Abständen speichern

4.1.2 Dienste allgemein

Jeder Dienst muss eine Reihe von Befehlen unterstützen. Um einen solchen Befehl auszuführen zu können muss der Benutzer in seinem Chatclient den Befehl als private Nachricht an den jeweiligen Dienst senden (siehe auch Kap. 2.3.3.5).

Dienste müssen ihre Antworten als NOTICE zurücksenden (siehe Kapitel 2.3.3.6).

Für sämtliche Anforderung die nachfolgend erfasst werden gilt für die Befehlssyntax:

<>: Optionale Parameter, Text

[]: Vordefinierter Parameter (Befehl)

4.1.3 Datenbank

Um die anfallenden Daten zu speichern benötigen wir eine Datenbank. Um Abhängigkeiten zu vermeiden verwenden wir eine SQLITE Datenbank, da keine separate Software installiert werden muss um diese zusammen mit dieser Applikation zu verwenden. Die Datenbank soll alle 20 Minuten automatisch gespeichert werden.

4.1.4 Logging

Sämtlicher Datenverkehr der über den Basis-Server läuft soll geloggt werden. Zusätzlich sollen die Start und Ende sämtlicher Datenbankfunktionen geloggt werden um das debuggen zu erleichtert. Ebenso sollen sämtliche Sqlite Fehlermeldungen in das Log geschrieben werden. Es ist daher eine Funktion zu bauen, mit der man einen Logeintrag der Kategorie DEBUG, WARN oder ERROR geschrieben werden kann.

4.2 Nickserv

Wir definieren die Stakeholder für Nickserv wie folgt:

Stakeholder	Beschreibung
Server Administrator	Kann Konfigurationen für Nickserv vornehmen und hat Vollzugriff auf alle Befehle. Das bedeutet, dass er Passwörter für beliebige Nicknames einsehen und neu setzen kann.
Benutzer	Kann Nickserv nutzen. Einige Befehle erfordern keine Berechtigung, die meisten jedoch schon.

Tabelle 10: Stakeholder Nickserv

Als weiteren Akteur definieren wir:

Akteur	Beschreibung
Nickserv	Empfängt Befehle von Nutzer und Administratoren, verarbeitet diese und liefert immer eine Antwort zurück.

Tabelle 11: Akteure Nickserv

4.2.1 Use Case Nickserv (UC NS-01)

Use Case Nickserv

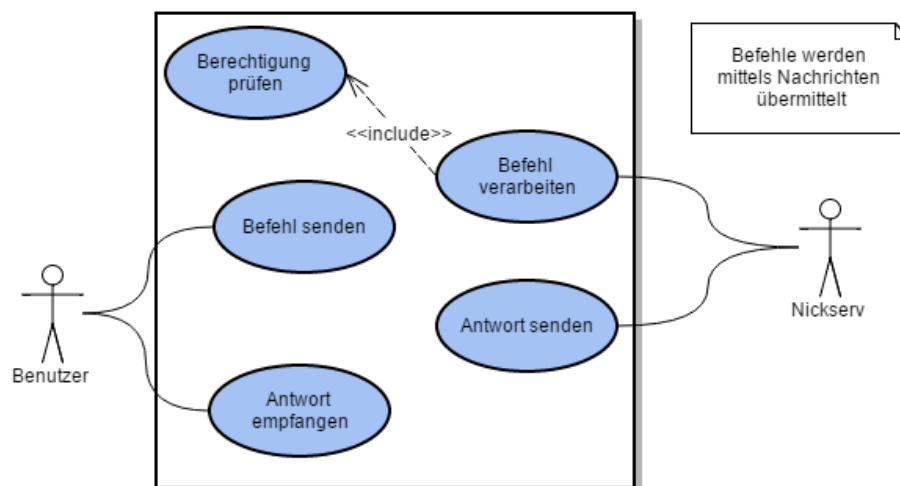


Abb. 7: Use Case Nickserv

4.2.2 Anforderungen Nickserv

Folgende Funktionen soll Nickserv anbieten

- **ACC** – Feststellung mit welcher Berechtigung ein Benutzer einen Nickname verwendet.
- **ACCESS** – Eine Liste in welche Benutzeradressen eingetragen werden können, sodass der Benutzer sich nicht explizit identifizieren muss, um den Nickname zu verwenden.
- **AUTH** – Ein Mechanismus mit der ein Benutzer explizit zustimmen muss, um zu Operator oder Notifylisten hinzugefügt zu werden.
- **DROP** – Registrierung eines Nickname aufheben
- **GETPASS** – Passwort eines Nicknames einsehen
- **GHOST** – Im Falle eines Ping-Timeout, also den Verlust der Verbindung kann es vorkommen, dass der Nickname trotzdem noch online ist (sogenannter Ghost). Diese Verbindung soll getrennt werden können.

- **IDENTIFY** – Identifikation für einen Nickname
- **INFO** – Zeigt Information über einen Nickname an
- **LIST** – Zeigt eine Liste von registrierten Nicknames an.
- **LISTCHANS** – Zeigt eine Liste von Channels in denen ein Nickname spezielle Rechte hat.
- **NOTIFY** – Eine Liste von „Freunden“. Verbindet sich ein Freund zum Server, sendet Nickserv eine Information.
- **REGISTER** – Einen Nickname registrieren.
- **RELEASE** – Wird ein Nickname ohne Berechtigung verwendet, so sperrt Nickserv die weitere Verwendung. Mit RELEASE kann diese Sperre aufgehoben werden.
- **SET** – Einstellungen für den Nickname. Diese sind:
 - **AUTHORIZE** – Stellt ein, ob und für welche Liste die explizite Zustimmung gegeben werden muss.
 - **EMAIL** – Legt die E-Mail Adresse fest
 - **HIDEEMAIL** – Legt fest, ob die E-Mail Adresse in der Nick Information angezeigt wird.
 - **MFORWARD** – Legt eine automatische Weiterleitung von Memos fest (Memoserv noch nicht implementiert)
 - **MLOCK** – Legt Benutzermodi fest, die automatisch nach der Identifikation gesetzt werden.
 - **NOMEMO** – Legt fest, ob ein Benutzer Memos erhalten will (Memoserv noch nicht implementiert)
 - **NOOP** – Legt fest, ob der Benutzer in Channel in denen er Operatoren-Rechte hat, automatisch diesen Status erhalten soll.
 - **PASSWORD** – Legt ein neues Passwort für den Nickname fest.
 - **PROTECT** – Legt den Schutz für den Nickname fest. Es stehen drei Schutz-Stufen zur Verfügung
 - **URL** – Legt eine URL fest, die in der Nick Info angezeigt wird.
- **SETPASS** – Legt ein neues Passwort für einen Nickname fest.

Die Funktionen müssen die nachfolgend definierten Fähigkeiten aufweisen.

4.2.3 Anforderung Nickserv Hauptfunktion

Requirement Nr.	R-NS-001
Titel	Nickserv Hauptfunktion
Beschreibung	Es ist Handler für private Nachrichten an Nickserv zu implementieren, welche als PRIVMSG an Nickserv gesendet werden. Die Nachricht muss im Format <code>source PRIVMSG Nickserv :Nachricht</code> vorliegen. Nickserv muss dann anhand einer dynamischen Befehlsliste automatisch aufgrund der Nachricht den korrekten Befehl aufrufen. Beinhaltet Nachricht keinen gültigen Befehl wird eine entsprechende Fehlermeldung zurückgegeben.
Kommentare	

Tabelle 12: R-NS-001: Nickserv Hauptfunktion

4.2.4 Konfiguration Nickserv

Requirement Nr.	R-NS-002
Titel	Konfiguration Nickserv
Beschreibung	Folgende Konfigurationsmöglichkeiten soll Nickserv anbieten: Abschnitt „general“

	<p>enabled: Nickserv aktivieren name: Nickname von Nickserv realname: Realname von Nickserv expiry: Zeit in Tagen wann der Nickname bei Nichtbenutzung gelöscht wird admin: Welchen Administrator Level ein Benutzer für den Vollzugriff benötigt. Diese sind: 1: Help Operator 2: IRC Operator 3: Co Administrator 4: Server Administrator 5: Services Administrator 6: Network Administrator</p> <p>Abschnitt „registration“ delay: Wie viele Sekunden vergangen sein müssen, bevor erneut ein Nickname registriert werden kann access: Welchen IRC Operator Level ein Benutzer haben muss, um einen Nickname zu registrieren. autoaccess: Legt fest, ob die Adresse des Benutzers automatisch in die ACCESS Liste des Nickname eingetragen werden soll.</p> <p>Abschnitt „list“ maxlist: Max. Anzahl Einträge die beim LIST Befehl angezeigt werden soll operonly: Nur IRC Operatoren können diesen Befehl nutzen</p> <p>Abschnitt „password“</p> <p>getpass: Benötigter Level um diesen Befehl zu nutzen setpass: Benötigter Level um diesen Befehl zu nutzen enforcer: Benutzernamen bei Sperre eines Nickname release: Zeit in Sekunden bevor ein gesperrter Nickname wieder freigegeben werden soll</p> <p>Abschnitt „default“</p> <p>Legt die Standardeinstellungen fest, die bei der Registrierung eines Nickname gesetzt werden sollen:</p> <p>noop: Nickname soll nicht automatisch Operator Status erhalten. high_protect: Setzt die höchste Schutzstufe hide_email: E-Mail wird in Info nicht angezeigt no_memo: Benutzer erhält keine Memos auth_channel: Benutzer muss zustimmen bevor er auf Operator Listen gesetzt werden kann. auth_notify: Benutzer muss zustimmen, bevor er auf Notify listen gesetzt werden kann. mnotify: Erhält eine Nachricht wenn ein neues Memo eintrifft mlock: Benutzermodi die bei der Identifizierung gesetzt werden sollen.</p>
Kommentare	

Tabelle 13: R-NS-002: Konfiguration Nickserv

4.2.5 Anforderungen Nickserv Unterfunktionen

4.2.5.1 Anforderungen ACC

Requirement Nr.	R-NS-003
Titel	ACC
Beschreibung	<p>Es soll ein Befehl implementiert der dem Absender die Berechtigung, die der angegebene Nutzer für den verwendeten Nickname hat, zurückgibt. Folgendes soll zurückgegeben werden:</p> <p>Hinweis, dass der Nickname nicht registriert ist, falls zutreffend</p> <p>0, falls der Nutzer keinerlei Zugriff auf den Nickname hat</p> <p>1, falls der Nutzer einen passenden Eintrag in der ACCESS List des Nickname hat</p> <p>2, falls der Nutzer sich mittels Passwort für den Nickname identifiziert hat.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ACC <Nickname></p>
Kommentare	<p>Keine besonderen Rechte erforderlich</p> <p>Implementierung von ACCESS ist erforderlich.</p>

Tabelle 14: R-NS-003: ACC

4.2.5.2 Anforderungen ACCESS

Requirement Nr.	R-NS-004
Titel	ACCESS
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Einträge von Benutzeradressen verwaltet werden können. Diese Benutzeradressen sollen den entsprechenden Nickname verwenden können, ohne dass sich der Benutzer explizit für den Nickname identifizieren muss.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu</p> <p>DEL – Entfernt einen Eintrag aus der Liste</p> <p>LIST – Zeigt alle Einträge der Liste an</p> <p>WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ACCESS [Befehl] <Adresse Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Um den Parameter <Nickname> zu nutzen muss der Benutzer Vollzugriff auf Nickserv haben.</p>

Tabelle 15: R-NS-004: ACCESS

4.2.5.3 Anforderungen AUTH

Requirement Nr.	R-NS-005
Titel	AUTH
Beschreibung	<p>Es soll ein Mechanismus geschaffen werden, mit der ein Benutzer seine Zustimmung geben kann, bevor er auf Listen gesetzt werden kann.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ACCEPT: Bestätigen eine Anfrage DECLINE: Ablehnen einer Anfrage READ: Eine Anfrage lesen LIST: Anfragen anzeigen.</p> <p>Die Befehlssyntax wird wie folgt definiert: ACCESS [Befehl] <Adresse Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>AUTHORIZE muss aktiviert sein. .</p>

Tabelle 16: R-NS-005: AUTH

4.2.5.4 Anforderungen DROP

Requirement Nr.	R-NS-006
Titel	DROP
Beschreibung	<p>Ein Nickname soll gelöscht werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert: DROP <Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Benutzer mit Vollzugriff können jeden Nickname löschen.</p>

Tabelle 17: R-NS-006: DROP

4.2.5.4 Anforderungen GETPASS

Requirement Nr.	R-NS-007
Titel	GETPASS
Beschreibung	<p>Administratoren sollen das Passwort eines Benutzers einsehen können.</p> <p>Die Befehlssyntax wird wie folgt definiert: GETPASS <Nickname></p>
Kommentare	<p>Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben</p>

Tabelle 18: R-NS-007: GETPASS

4.2.5.5 Anforderungen GHOST

Requirement Nr.	R-NS-008
Titel	GHOST
Beschreibung	<p>Verliert ein Benutzer die Verbindung kann es vorkommen, das ein Abbild der Verbindung immer noch auf dem Server ist. Diese Verbindung soll getrennt werden können.</p>

	Die Befehlssyntax wird wie folgt definiert: GHOST <Nickname> <Passwort>
Kommentare	Benutzer muss für den Nickname identifiziert sein.

Tabelle 19: R-NS-008: GHOST

4.2.5.6 Anforderungen IDENTIFY

Requirement Nr.	R-NS-009
Titel	IDENTIFY
Beschreibung	Ein Benutzer muss sich für seinen Nickname identifizieren können. Wenn ein Benutzer das Passwort dreimal falsch eingibt, soll eine vorkonfigurierte Aktion ausgeführt werden. Die Befehlssyntax wird wie folgt definiert: IDENTIFY<Nickname> <Passwort>
Kommentare	Diese Funktion kann von jedem Benutzer verwendet werden.

Tabelle 20: R-NS-009: IDENTIFY

4.2.5.7 Anforderungen INFO

Requirement Nr.	R-NS-010
Titel	INFO
Beschreibung	Es sollen grundlegende Informationen über einen Nickname angezeigt werden können. Die Befehlssyntax wird wie folgt definiert: INFO <Nickname>
Kommentare	Diese Funktion kann von jedem Benutzer verwendet werden.

Tabelle 21: R-NS-010: INFO

4.2.5.8 Anforderungen LIST

Requirement Nr.	R-NS-011
Titel	LIST
Beschreibung	Die registrierten Nicknamen sollen in einer Liste angezeigt werden können. Es soll eine Suchmaske als Argument übergeben werden können. Die Befehlssyntax wird wie folgt definiert: LIST <Suchmaske>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 22: R-NS-011: LIST

4.2.5.9 Anforderungen LISTCHANS

Requirement Nr.	R-NS-012
Titel	LISTCHANS

Beschreibung	<p>Der Benutzer soll die Möglichkeit haben, alle Channels anzuzeigen in denen er erweiterte Rechte hat. Anzuzeigen sind Channel, Zugriff, wer den Zugriff hinzugefügt hat und das Datum.</p> <p>Die Befehlssyntax wird wie folgt definiert: LISTCHANS <Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Benutzer mit Vollzugriff können die Liste für jeden Nickname anzeigen.</p>

Tabelle 23: R-NS-012: LISTCHANS

4.2.5.10 Anforderungen NOTIFY

Requirement Nr.	R-NS-013
Titel	NOTIFY
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Freunde verwaltet werden können. Wenn ein Freund sich zum Server verbindet sendet Nickserv eine Nachricht an den Benutzer.</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Befehlssyntax wird wie folgt definiert: NOTIFY [Befehl] < Nickname></p>
Kommentare	Der Benutzer muss für den Nickname identifiziert sein

Tabelle 24: R-NS-013: NOTIFY

4.2.5.11 Anforderungen REGISTER

Requirement Nr.	R-NS-014
Titel	REGISTER
Beschreibung	<p>Um einen Nickname zu reservieren und vor unbefugtem Zugriff zu schützen soll dieser registriert werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert: REGISTER <Passwort> <E-Mail Adresse></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 25: R-NS-014: REGISTER

4.2.5.12 Anforderungen RELEASE

Requirement Nr.	R-NS-015
Titel	RELEASE
Beschreibung	<p>Wird ein Nickname gesperrt, so muss der rechtmässige Benutzer die Möglichkeit haben die Sperre wieder aufzuheben.</p> <p>Die Befehlssyntax wird wie folgt definiert: RELEASE <Nickname> <Passwort></p>

Kommentare	
------------	--

Tabelle 26: R-NS-015: RELEASE

4.2.5.13 Anforderungen SET

Requirement Nr.	R-NS-015
Titel	SET
Beschreibung	<p>Der Benutzer soll die Möglichkeit haben, Einstellungen vorzunehmen:</p> <p>AUTHORIZE – Stellt ein, ob und für welche Liste die explizite Zustimmung gegeben werden muss.</p> <p>EMAIL – Legt die E-Mail Adresse fest</p> <p>HIDEEMAIL – E-Mail Adresse soll in der Nick Info angezeigt werden</p> <p>MFORWARD – Legt eine automatische Weiterleitung von Memos fest</p> <p>MLOCK – Legt Modi fest, die nach der Identifikation gesetzt werden.</p> <p>NOMEMO – Legt fest, ob ein Benutzer Memos erhalten will</p> <p>NOOP – Legt fest, ob der Benutzer in Channel in denen er Operatoren-Rechte hat, automatisch diesen Status erhalten soll.</p> <p>PASSWORD – Legt ein neues Passwort für den Nickname fest.</p> <p>PROTECT – Legt den Schutz für den Nickname fest. Es stehen drei Schutz-Stufen zur Verfügung</p> <p>URL: URL, die in der Nick Information angezeigt werden soll.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SET [Befehl] <Einstellung></p>
Kommentare	Der Benutzer muss für den Nickname identifiziert sein

Tabelle 27: R-NS-016: SET

4.2.5.14 Anforderungen SETPASS

Requirement Nr.	R-NS-016
Titel	SETPASS
Beschreibung	<p>Administratoren sollen das Passwort eines Benutzers neu setzen können</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SETPASS <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 28: R-NS-016: SETPASS

4.2.6 Weitere Nickserv Anforderungen

4.2.6.1 Identifikationstimer

Requirement Nr.	R-NS-017
Titel	Identifikationstimer
Beschreibung	Wird der Schutz eines Nicknames auf Normal gestellt, soll der Benutzer 60 Sekunden Zeit haben, sich zu identifizieren. Nach 30 Sekunden soll eine entsprechende Warnung ausgegeben werden.
Kommentare	

Tabelle 29: R-NS-017: Identifikationstimer

4.3 Chanserv

4.3.1 Use Case Chanserv (UC CS-01)

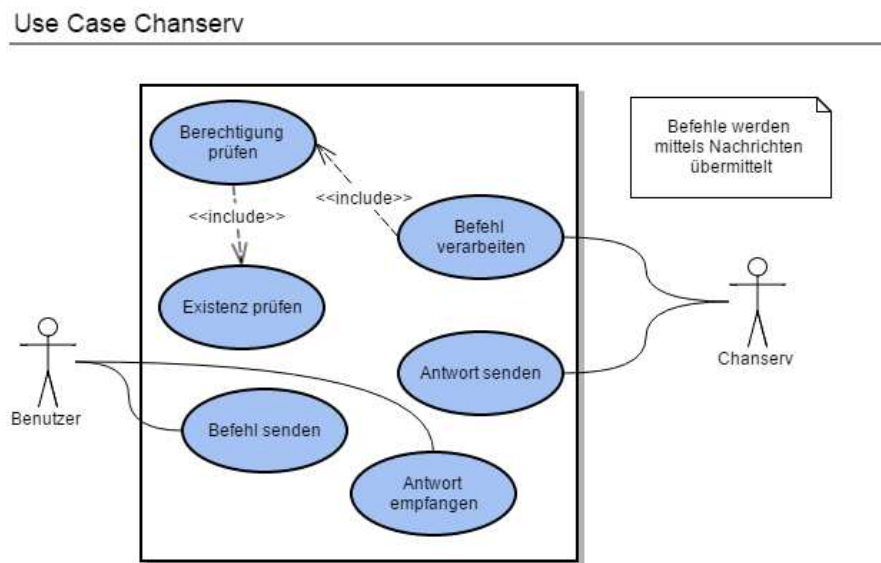


Abb. 8: Use Case Chanserv

4.3.2 Anforderungen Chanserv

Folgende Funktionen soll Chanserv anbieten

- **ACC** – Feststellung mit welcher Berechtigung ein Benutzer einen Nickname verwendet.
- **AKICK** – Eine Liste von Benutzeradressen, die automatisch aus dem Channel „gekickt“ werden.
- **AOP** – Steht für Auto Operator. Benutzer in dieser Liste haben Operatoren-Berechtigungen für den Channel.
- **DEHALFOP** – Entfernt den Half-Operatoren Status eines Benutzers
- **DEVOICE** – Entfernt den Voice Status eines Benutzers
- **DROP** – Löscht einen Channel
- **GETPASS** – Zeigt das Passwort des Channels an.
- **HALFOP** – Gibt einem Benutzer Half-Operatoren Status
- **HOP** – Steht für Half-Operator. Ein Half-Operator hat eingeschränkte Operatoren-Rechte.
- **IDENTIFY** – Identifikation für einen Channel
- **INFO** – Zeigt Information über einen Channel an
- **INVITE** – Ermöglicht es einem Benutzer der mindestens User Operator (Uop) Recht in einem Channel hat, sich selber in einen Channel einzuladen, falls nur eingeladene Benutzer den Channel betreten dürfen.
- **LIST** – Zeigt eine Liste von registrierten Channel an.
- **MKICK** – Kickt alle Benutzer aus einem Channel.
- **MDEOP** – Entfernt den Operatorenstatus von allen Operatoren in einem Channel.
- **OP** – Gibt einem Benutzer den Operatorenstatus in einem Channel.
- **REGISTER** – Einen Channel registrieren.
- **SET** – Einstellungen für den Channel. Diese sind:
 - **BOT** – Legt einen existierenden Bot fest, der für den Channel zuständig ist und diesen offen hält.

- **DESC** – Legt die Beschreibung des Channel fest.
- **FOUNDER** – Legt den Gründer des Channels fest. Muss ein registrierter Nickname sein.
- **OPWATCH** – Legt einen höheren Schutz für den Channel fest. Es können Benutzer mit explizitem Zugriff (Aop oder höher) Operatoren-Rechte gegeben werden.
- **LEAVEOPS** – Legt einen tieferen Schutz für den Channel fest. Man kann grundsätzlich jedem Benutzer Operatoren-Rechte geben.
- **KEEPTOPIC** – Legt fest, ob das Topic (Titel des Channels) beim Schliessen des Channels gemerkt und bei der erneuten Öffnung dessen neu gesetzt werden soll.
- **MEMOLEVEL** – Legt den Level fest, den ein Benutzer haben muss, um ein Memo an alle Benutzer mit Zugriff auf den Channel zu senden (Memoserv noch nicht implementiert)
- **MLOCK** – Legt Channelmodi fest, die nicht geändert werden können.
- **PASSWORD** – Legt ein neues Passwort für den Channel fest.
- **RESTRICTED** – Legt fest, ob nur Benutzer mit explizitem Zugriff (Uop oder höher) den Channel betreten dürfen.
- **SUCCESSOR** – Legt einen Nachfolger für den Channel fest. Wird automatisch als Founder gesetzt wenn der Nickname des Founders gelöscht wird.
- **TOPICLOCK** – Legt den Level fest, den ein Benutzer haben muss um das Topic des Channel zu ändern.
- **SETPASS** – Legt ein neues Passwort für einen Channel fest.
- **SOP** – Steht für Super Operator. Benutzer in dieser Liste haben Operatoren-Berechtigungen für den Channel und können andere Benutzer der Aop Liste (oder tiefer) hinzufügen.
- **UNBAN** – Löscht einen Ban, der auf einen Benutzer zutrifft, der Zugriff auf den Channel hat.
- **UOP** – Steht für User Operator. Benutzer in dieser Liste haben keinerlei Rechte, ausser, dass sie als Benutzer mit Zugriff auf den Channel gelten.
- **VOICE** – Gibt einem Benutzer Voice Status in einem Channel.
- **VOP** – Steht für Voice Operator. Benutzer in dieser Liste haben Voice-Berechtigungen für den Channel. Das heisst, dass sie in einem moderierten Channel (Channel Modus „voice only“ Nachrichten an den Channel senden können.

4.3.3 Anforderung Chanserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Chanserv sind identisch mit jenen von Nickserv.

4.3.4 Konfiguration Chanserv

Requirement Nr.	R-CS-001
Titel	Konfiguration Chanserv
Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Chanserv anbieten:</p> <p>Abschnitt „general“ enabled: Chanserv aktivieren name: Nickname von Chanserv realname: Realname von Chanserv</p> <p>Abschnitt „settings“: Welchen Zugriffslevel ein Benutzer für die Einstellungen der folgenden Einstellungen benötigt. Die Level sind wie folgt definiert: 6: Successor: Nickname hat Successor Level im Channel</p>

	<p>7: Der Benutzer hat sich für den Founder (Gründer) Nickname identifiziert.</p> <p>8: Der Benutzer hat sich für über Chanserv für den Channel identifiziert.</p> <p>Abschnitt „access“ Dieser Abschnitt beschreibt den benötigten Zugriffslevel um den entsprechenden Befehl auszuführen. Diese sind: 1 Uop 2 Vop 3 Hop 4 Aop 5 Sop 6 Successor 7 Für den Founder Nickname identifiziert 8 Über Chanserv für den Channel identifiziert. Folgende Unterabschnitte und Befehle müssen konfigurierbar sein:</p> <ul style="list-style-type: none"> • Abschnitt akick mit den Befehlen add, del, list, wipe • Abschnitt sop mit den Befehlen add, del, list, wipe • Abschnitt aop mit den Befehlen add, del, list, wipe • Abschnitt hop mit den Befehlen add, del, list, wipe • Abschnitt vop mit den Befehlen add, del, list, wipe • Abschnitt uop mit den Befehlen add, del, list, wipe <p>Sowie die Befehle mkick und mdeop.</p> <p>Bei den Zugriffskonfigurationen muss dabei eine Validierung vorgenommen werden, dass die Zugriffe Sinn ergeben. Das heisst zum Beispiel, dass ein Aop keine Sop hinzufügen können darf.</p> <p>Abschnitt „list“ maxlist: Max. Anzahl Einträge die beim LIST Befehl angezeigt werden soll operonly: Nur IRC Operatoren können diesen Befehl nutzen</p> <p>Abschnitt „registration“ delay: Wieviele Sekunden vergangen sein müssen, bevor erneut ein Nickname registriert werden kann</p> <p>Abschnitt „password“ getpass: Benötigter Level um diesen Befehl zu nutzen setpass: Benötigter Level um diesen Befehl zu nutzen</p> <p>Abschnitt „default“ Legt die Standardeinstellungen fest, die bei der Registrierung eines Channels gesetzt werden sollen:</p> <p>keeptopic: Topic soll beim Schliessen des Channel gemerkt werden leaveops: Operatoren ohne Zugriff sollen erlaubt sein</p>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	memolevel: Benötigter Zugriffslevel um Memos an Benutzer des Channels zu senden opwatch: Operatoren ohne Zugriff sollen nicht erlaubt sein restricted: Nur Benutzer mit Zugriff sollen den Channel betreten können topiclock: Benötigter Zugriffslevel um das Topic des Channels zu ändern autovop: Benutzer mit Vop Zugriff erhalten automatisch den Voice Status mlock: Channelmodi die nicht geändert werden können.
Kommentare	

Tabelle 30: R-CS-002: Konfiguration Chanserv

4.3.5 Anforderungen Chanserv Unterfunktionen

4.3.5.1 ACC

Requirement Nr.	R-CS-002
Titel	ACC
Beschreibung	<p>Es soll ein Befehles namens ACC implementiert der dem Absender die Berechtigung, die der angegebene Nutzer für den angegebenen Channel hat, zurückgibt.</p> <p>Folgendes soll zurückgegeben werden:</p> <p>Zugriffslevel Wieso der Benutzer diesen Zugriff hat.</p> <p>Beispiel Benutzer B hat Aop Zugriff auf Channel C, weil er sich für den Nickname A identifiziert hat</p> <p>Die Befehlssyntax wird wie folgt definiert: ACC <Channel> <Nickname> </p>
Kommentare	Mindestens Uop Zugriff ist erforderlich.

Tabelle 31: R-CS-002: ACC

4.3.5.2 Anforderungen AKICK

Requirement Nr.	R-CS-003
Titel	AKICK
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Einträge von Benutzeradressen in einem Channel verwaltet werden können, für die Benutzer zu denen diese Adresse gehören automatisch aus dem Channel gekickt werden.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Maske muss validiert werden und folgendes Format aufweisen:</p> <p>Nickname!Benutzer@host.tld</p>

	<p>Sterne als Platzhalter werden akzeptiert. Beispiele:</p> <p>Nick!user@host *!*@host</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AKICK <Channel> [Befehl] <Adresse ></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 32: R-CS-003: AKICK

4.3.5.3 Anforderungen AOP

Requirement Nr.	R-CS-004
Titel	AOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Auto Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Operatoren-Rechte geben und die konfigurierten Befehle ausführen.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 33: R-CS-004: AOP

4.3.5.4 Anforderungen DE-/HALFOP

Requirement Nr.	R-CS-005
Titel	HALFOP / DEHALFOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein Nickname, der Hop Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>HALFOP <Channel> <Nickname> DEHALFOP <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 34: R-CS-005: DE-/HALFOP

4.3.5.5 Anforderungen DE-/OP

Requirement Nr.	R-CS-006
Titel	DE-/OP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein Nickname, der Operatoren Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>OP <Channel> <Nickname> DEOP <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 35: R-CS-006: DE-/OP

4.3.5.6 Anforderungen DE-/VOICE

Requirement Nr.	R-CS-007
Titel	DE-/VOICE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein Nickname, der Vop Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOICE <Channel> <Nickname> DEVOICE <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 36: R-CS-007: DEVOICE

4.3.5.7 Anforderungen DROP

Requirement Nr.	R-CS-008
Titel	DROP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein registrierter Channel gelöscht werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>DROP <Channel></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 37: R-CS-008: DROP

4.3.5.8 Anforderungen GETPASS

Requirement Nr.	R-CS-009
Titel	GETPASS

Beschreibung	Administratoren sollen das Passwort eines Channel einsehen können. Wird dieser Befehl benutzt muss ein Logeintrag geschrieben werden. Die Befehlssyntax wird wie folgt definiert: GETPASS <Channel>
Kommentare	Der Benutzer benötigt die konfigurierten Zugriffsrechte für diesen Befehl.

Tabelle 38: R-CS-009: GETPASS

4.3.5.9 Anforderungen HOP

Requirement Nr.	R-CS-010
Titel	HOP
Beschreibung	Es soll eine Liste geschaffen werden, mit der Half Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Half-Operatoren-Rechte geben und die konfigurierten Befehle ausführen. Es sollen folgende Unterfunktionen geschaffen werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen. Die Befehlssyntax wird wie folgt definiert: HOP <Channel> [Befehl] <Nickname>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 39: R-CS-010: HOP

4.3.5.10 Anforderungen IDENTIFY

Requirement Nr.	R-CS-011
Titel	IDENTIFY
Beschreibung	Es soll ein Mechanismus geschaffen werden, mit der ein Benutzer sich für den Channel identifizieren kann. Wenn der Benutzer sich identifiziert hat, erhält er den Vollzugriff für den Channel. Die Befehlssyntax wird wie folgt definiert: chanserv IDENTIFY<Channel> <Passwort>
Kommentare	

Tabelle 40: R-CS-011: IDENTIFY

4.3.5.11 Anforderungen INVITE

Requirement Nr.	R-CS-012
Titel	INVITE

Beschreibung	<p>Da es möglich ist, dass Channels nur Einladung betreten werden können muss ein Befehl geschaffen werden, mit der Benutzer die Zugriff auf den Channel haben sich über Chanserv selber einladen können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>INVITE <Channel></p>
Kommentare	Mindestens Uop Zugriff benötigt.

Tabelle 41: R-CS-013: INVITE

4.3.5.12 Anforderungen LIST

Requirement Nr.	R-CS-013
Titel	LIST
Beschreibung	<p>Die registrierten Channels sollen in einer Liste angezeigt werden können. Es soll eine Suchmaske als Argument übergeben werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LIST <Suchmaske></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 42: R-CS-013: LIST

4.3.5.13 Anforderungen MDEOP

Requirement Nr.	R-CS-014
Titel	MDEOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit welchem sämtlichen Operatoren eines Channel der Status entzogen werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MDEOP <Channel></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 43: R-CS-014: MDEOP

4.3.5.14 Anforderungen MKICK

Requirement Nr.	R-CS-015
Titel	MKICK
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit welchem sämtliche Benutzer aus einem Channel gekickt werden können. Wird dieser Befehl ausgeführt soll Chanserv automatisch den Channel vorübergehend betreten damit dieser offen bleibt.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MKICK <Channel> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 44: R-CS-015: MKICK

4.3.5.15 Anforderungen REGISTER

Requirement Nr.	R-CS-016
Titel	REGISTER
Beschreibung	<p>Soll ein Befehl geschaffen werden, mit dem man einen Channel mit Chan-serv registrieren kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>REGISTER <Channel> <Passwort> <Beschreibung></p>
Kommentare	<p>Der Nutzer muss einen registrierten Nickname haben.</p> <p>Der Nutzer muss die konfigurierten Rechte für die Registrierung von Channels haben</p>

Tabelle 45: R-CS-016: REGISTER

4.3.5.16 Anforderungen SET

Requirement Nr.	R-CS-017
Titel	SET
Beschreibung	<p>Der Gründer eines Channels soll die Möglichkeit haben, Einstellungen vorzunehmen:</p> <p>BOT – Legt einen Bot für den Channel fest. Der Bot muss über Botserv registriert werden.</p> <p>DESCRIPTION – Legt die Beschreibung des Channels fest</p> <p>FOUNDER – Legt einen neuen Gründer für den Channel fest. Der Gründer muss einen registrierten Nickname haben.</p> <p>KEEPTOPIC – Legt fest, ob das Topic beim Schliessen des Channel gemerkt werden soll.</p> <p>LEAVEOPS – Legt fest, ob Benutzer ohne Aop Zugriff oder höher Op-Rechte erhalten können. Wenn diese Option aktiviert wird soll Opwatch automatisch deaktiviert werden.</p> <p>MEMOLEVEL – Legt fest, welchen Zugriff ein Benutzer auf den Channel haben muss, um Memos an alle Benutzer mit Zugriff zu schicken.</p> <p>MLOCK – Legt Modi fest, die nicht verändert werden können.</p> <p>OPWATCH – Legt fest, ob nur Benutzer mit entsprechender Berechtigung Op Rechte erhalten kann. Wenn diese aktiviert wird muss Leaveops automatisch deaktiviert werden.</p> <p>PASSWORD – Legt ein neues Passwort für den Nickname fest.</p> <p>RESTRICTED – Nur Benutzer mit Uop Zugriff oder höher dürfen den Channel betreten.</p> <p>SUCCESSOR – Legt einen Nachfolger für den Channel fest. Wenn der Nickname des Founders gelöscht wird soll automatisch der Nachfolger als Founder gesetzt werden. Der Nachfolger muss einen registrierten Nickname haben</p> <p>TOPICLOCK – Legt fest, welchen Zugriff ein Benutzer auf den Channel haben muss, um das Topic des Channels zu setzen. Mit OFF soll jeder Op das Topic ändern können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SET <Channel> [Befehl] <Einstellung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte haben.

Tabelle 46: R-CS-017: SET

4.3.5.17 Anforderungen SETPASS

Requirement Nr.	R-CS-018
Titel	SETPASS
Beschreibung	Administratoren sollen das Passwort eines Channel neu setzen können. Wird dieser Befehl benutzt muss ein Logeintrag geschrieben werden. Die Befehlssyntax wird wie folgt definiert: SETPASS <Channel> <Passwort>
Kommentare	Der Benutzer benötigt die konfigurierten Zugriffsrechte für diesen Befehl.

Tabelle 47: R-CS-018: SETPASS

4.3.5.18 Anforderungen SOP

Requirement Nr.	R-CS-019
Titel	SOP
Beschreibung	Es soll eine Liste geschaffen werden, mit der Super Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Operatoren-Rechte geben und die konfigurierten Befehle ausführen. Es sollen folgende Unterfunktionen geschaffen werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste Es dürfen nur registrierte Nicknames der verwendet werden. Dies zu prüfen. Die Befehlssyntax wird wie folgt definiert: SOP <Channel> [Befehl] <Nickname>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 48: R-CS-019: SOP

4.3.5.19 Anforderungen UNBAN

Requirement Nr.	R-CS-020
Titel	UNBAN
Beschreibung	Es soll ein Befehl geschaffen werden mit dem Benutzer die Uop Zugriff oder höher auf einen Channel haben, allfällige Bans aus dem Channel entfernen können. Alle Bans die auf den Benutzer zutreffen sollen gefunden und gelöscht werden. Die Befehlssyntax wird wie folgt definiert: UNBAN<Channel>
Kommentare	Der Benutzer benötigt Uop Zugriff oder höher

Tabelle 49: R-CS-020: UNBAN

4.3.5.20 Anforderungen UOP

Requirement Nr.	R-CS-021
Titel	UOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der User Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff gelten als Benutzer mit Zugriff auf den Channel, haben aber sonst keine speziellen Rechte. Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>UOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 50: R-CS-021: UOP

4.3.5.21 Anforderungen VOP

Requirement Nr.	R-CS-022
Titel	UOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Voice Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich mit dem VOICE Befehl selber den Status im Channel geben. Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 51: R-CS-022: Uop

4.4 Operserv

4.4.1 Use Case Operserv (UC OS-01)

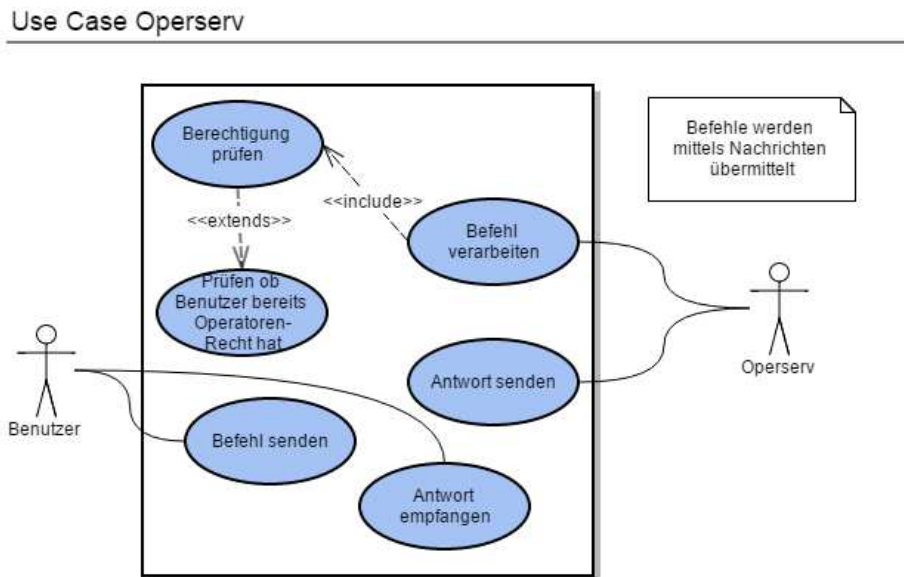


Abb. 9: Use Case Operserv

4.4.2 Anforderungen Operserv

Folgende Funktionen soll Operserv anbieten

- **AKILL** – Hinzufügen eines AUTOKILLS
- **CHATOPS** – Nachricht an alle Server Operatoren
- **CHGHOST** – Ändern des Hostnamen eines Benutzers
- **GLOBAL**–Nachricht an alle Server Operatoren auf dem Netzwerk
- **KILL**– Trennen der Verbindung eines Benutzers
- **LOCAL**–Nachricht an alle Server Operatoren auf dem Server, jedoch nicht in gesamten Netzwerk
- **OPER**– Verwalten der Operatoren-Liste
- **SGLINE** – Service G:Line. Dies bedeutet einen Ban auf dem gesamtem Netzwerk auf Basis der Hostmaske
- **SKLINE**– Service K:Line. Dies bedeutet einen Ban auf dem Lokalen Server auf Basis der Hostmaske
- **SQLINE** – Service Q:Line. Dies verhindert die Benutzung eines spezifischen Nickname
- **SZLINE**– Service Z:Line. Dies bedeutet einen Ban auf dem gesamten Netzwerk auf Basis der IP-Adresse. Der Unterschied zur G:Line ist dabei, dass ganze IP-Adressen Bereich gesperrt werden können.

4.4.3 Anforderung Operserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Operserv sind identisch mit jenen von Nickserv.

4.4.4 Konfiguration Operserv

Requirement Nr.	R-OS-001
Titel	Konfiguration Operserv
Beschreibung	Folgende Konfigurationsmöglichkeiten soll Operserv anbieten:

	<p>Abschnitt „general“ enabled: Opserv aktivieren name: Nickname von Opserv realname: Realname von Opserv access_flag: Welche Operatoren Zugriff ist nötig um Vollzugriff auf Opserv zu haben.</p> <p>Abschnitt „global“: Legt fest, bei welchen Events ein GLOBAL herausgeschickt werden soll, jeweils 0 oder 1 ist zulässig. Folgende Events sind konfigurierbar: on_oper, on_akill, on_sgline, on_skline, on_sqline, on_szline, on_kill</p> <p>Abschnitt „default“ Legt die Standard-Berechtigungen für einen neuen Oper fest. Jeweils 0 oder sind zulässig. Jeder Opserv Befehl ausser Oper ist zu berücksichtigen. Zusätzlich soll ein Standard Vhost (Virtual Host) gesetzt werden. Benutzer ohne Server Operatoren-Rechte aber mit Zugriff auf Opserv soll diesen erhalten.</p>
Kommentare	

Tabelle 52: R-OS-001: Konfiguration Opserv

4.4.5 Anforderungen Opserv Unterfunktionen

4.4.5.1 Anforderungen AKILL

Requirement Nr.	R-OS-002
Titel	AKILL
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit dem die Verbindung von störenden Benutzern automatisch getrennt werden kann. Dies kommt einem Ban auf den Server gleich.</p> <p>Folgende Unterbefehle sollen unterstützt werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an</p> <p>Als Zeitdauer sollen folgende Kürzel unterstützt werden:</p> <ul style="list-style-type: none"> • h – Stunden • w – Wochen • m – Monate • y – Jahre <p>Wird kein Kürzel angegeben, so ist die Zeitdauer in Minuten zu interpretieren. Die Dauer soll optional sein. Wird keine festgelegt so ist der Auto-kill permanent bis er manuell entfernt wird.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AKILL [Befehl] <Adresse > <Dauer> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 53: R-OS-002: AKILL

4.4.5.2 Anforderungen CHATOPS

Requirement Nr.	R-OS-003
Titel	CHATOPS
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine CHATOPS Nachricht an alle Operatoren gesendet werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>CHATOPS <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 54: R-OS-003: CHATOPS

4.4.5.3 Anforderungen CHGHOST

Requirement Nr.	R-OS-004
Titel	CHGHOST
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein CHGHOST Befehl abgesetzt werden kann. Dies soll den Host eines Benutzers ändern.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>CHGHOST <Benutzer> <Neuer Hostname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 55: R-OS-004: CHGHOST

4.4.5.4 Anforderungen GLOBAL

Requirement Nr.	R-OS-005
Titel	GLOBAL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein GLOBAL Befehl abgesetzt werden kann. So kann man eine Nachricht an alle Server Operatoren auf dem gesamten Netzwerk senden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>GLOBAL <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 56: R-OS-005: GLOBAL

4.4.5.5 Anforderungen KILL

Requirement Nr.	R-OS-006
Titel	KILL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem die Verbindung eines störenden Benutzers getrennt werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>KILL <Benutzer> <Begründung></p>

Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben
------------	------------------------------------------------------------------------------------

Tabelle 57: R-OS-006: KILL

4.4.5.6 Anforderungen LOCAL

Requirement Nr.	R-OS-007
Titel	GLOBAL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein LOCOPS Befehl abgesetzt werden kann. So kann man eine Nachricht an alle Server Operatoren auf dem aktuellen Server senden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LOCAL <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 58: R-OS-007: LOCAL

4.4.5.7 Anforderungen OPER

Requirement Nr.	R-OS-008
Titel	OPER
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem Benutzer, die noch über keine Operatoren-Rechte verfügen, solche gewähren kann. Der Zugriff auf die einzelnen Befehle soll individuell eingestellt werden können.</p> <p>Folgende Unterbefehle sollen unterstützt werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an SET – Legt die individuellen Einstellungen fest. Folgende Rechte können eingestellt werden:</p> <p>AKILL CHGHOST CHATOPS GLOBAL LOCAL KILL SGLINE SKLINE SQLINE SZLINE VHOST</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>OPER [Befehl] <Nickname> <Zugriff> [ENABLE DISABLE <Hostname>]</p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 59: R-OS-008: OPER

4.4.5.8 Anforderungen Server-Bans und Nicksperrn

Requirement Nr.	R-OS-009
Titel	SGLINE / SKLINE/ SZLINE / SQLINE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine neue G:Line, K:Line, Z:Line oder Q:Line gesetzt werden kann.</p> <p>Folgende Unterbefehle sollen unterstützt werden: ADD –Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste</p> <p>Als Zeitdauer sollen folgende Kürzel unterstützt werden:</p> <ul style="list-style-type: none"> • h – Stunden • w – Wochen • m – Monate • y – Jahre <p>Wird kein Kürzel angegeben, so ist die Zeitdauer in Minuten zu interpretieren. Die Dauer soll optional sein. Wird keine festgelegt so ist die G:LINE permanent bis sie manuell entfernt wird.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SGLINE [Befehl] <Adresse > <Dauer> <Begründung> SKLINE [Befehl] <Adresse > <Dauer> <Begründung> SZLINE [Befehl] <Adresse > <Dauer> <Begründung> SQLINE [Befehl] <Nickname> <Dauer> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 60: R-OS-009: SGLINE / SKLINE/ SZLINE / SQLINE

4.5 Botserv

4.5.1 Use Case Botserv (UC BS-01)

Use Case Botserv

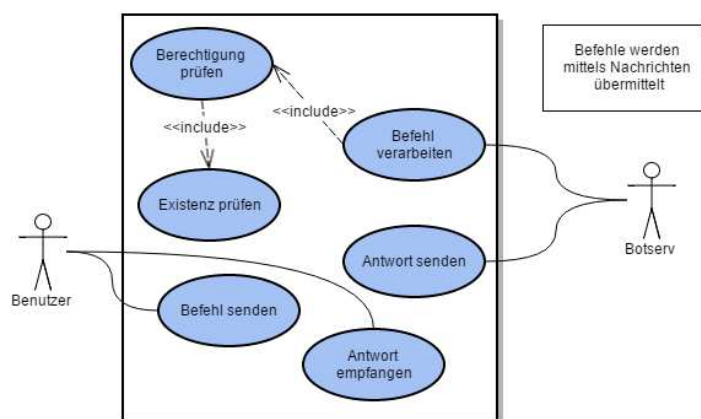


Abb. 10: Use Case Botserv

4.5.2 Anforderungen Botserv

Folgende Funktionen soll Botserv anbieten

- **ADD** – Hinzufügen eines Bots
- **DEL**– Entfernen eines Bots
- **DEHALFOP**– Entziehen des H
- **DEOP** – Entziehen des
- **DEVOICE**– Entziehen des
- **GETPASS**– Einsehen des Passwortes eines Bots. s
- **HALFOP**– Gewähren des Half-Operator Status
- **IDENTIFY**– Identifikation für den Bot
- **INFO**–Informationen über einen Bot anzeigen
- **KICK**– Kicken eines Benutzers aus dem Channel
- **LIST**– Liste der Bots anzeigen lassen.
- **MSG**– Nachricht über den Bot an einen Channel senden
- **OP** – Gewähren des Operator Status
- **SET** – Einstellungen am Bot vornehmen. Folgende Einstellungen können vorgenommen werden:
 - **NAME**: Neuer Nickname für den Bot
 - **PASSWORD**: Neues Passwort
 - **REALNAME**: Neuer Realname
 - **USERNAME**: Neuer Username
- **SETPASS**: Neues Passwort für den Bot setzen.
- **VOICE**– Gewähren des Voice Operator Status

4.5.3 Anforderung Botserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Botserv sind identisch mit jenen von Nickserv.

4.5.4 Konfiguration Botserv

Requirement Nr.	R-BS-001
Titel	Konfiguration Botserv
Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Botserv anbieten:</p> <p>Abschnitt „general“ enabled: Botserv aktivieren name: Nickname von Botserv realname: Realname von Botserv</p> <p>Abschnitt „access“: Legt fest, welchen Operatoren Zugriff ein Benutzer haben muss, um den jeweiligen Befehl auszuführen. Folgende Werte sind zulässig: 0: Jeder Benutzer darf den Befehl verwenden 1: Help Operatoren 2: IRC Operatoren 3: Co Admins 4: Server Admins 5: Services Admins 6: Network Admins</p>

	Folgende Befehle sollen konfigurieren werden können: add, del, list, set, getpass, setpass
Kommentare	

Tabelle 61: R-OS-001: Konfiguration Botserv

4.5.5 Anforderungen Botserv Unterfunktionen

4.5.5.1 Anforderungen ADD/DEL

Requirement Nr.	R-BS-002
Titel	ADD/DEL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein neuer Bot erstellt oder ein bestehender gelöscht werden kann. Beim Erstellen sollen der Nickname des Bots auch für den Username und den Realname gesetzt werden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ADD <Name> <Passwort> DEL <Name></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 62: R-BS-002: ADD/DEL

4.5.5.2 Anforderungen DE-/HALFOP

Requirement Nr.	R-BS-003
Titel	DE-/HALFOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Half-Operatoren Rechte geben oder entziehen kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>HALFOP <Bot> <Channel> <Nickname> DEHALFOP <Bot> <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 63: R-BS-003: DE-/HALFOP

4.5.5.3 Anforderungen DE-/VOICE

Requirement Nr.	R-BS-004
Titel	DE-/VOICE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Voice-Operatoren Rechte geben oder entziehen kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOICE <Bot> <Channel> <Nickname> DEVOICE <Bot> <Channel> <Nickname></p>

Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben
------------	------------------------------------------------------------------------------------

Tabelle 64: R-BS-004: DE-/VOICE

4.5.5.4 Anforderungen DE-/OP

Requirement Nr.	R-BS-005
Titel	DE-/OP
Beschreibung	Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Operatoren Rechte geben oder entziehen kann. Die Befehlssyntax wird wie folgt definiert: OP <Bot> <Channel> <Nickname> DEOP <Bot> <Channel> <Nickname>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 65: R-BS-005: DE-/OP

4.5.5.5 Anforderungen GETPASS

Requirement Nr.	R-BS-006
Titel	GETPASS
Beschreibung	Administratoren sollen das Passwort eines Bots einsehen können. Wird dieser Befehl verwendet muss ein Log-Eintrag geschrieben werden. Die Befehlssyntax wird wie folgt definiert: GETPASS<Bot>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 66: R-BS-006: GETPASS

4.5.5.6 Anforderungen IDENTIFY

Requirement Nr.	R-BS-007
Titel	IDENTIFY
Beschreibung	Es soll ein Befehl geschaffen werden mit dem ein Benutzer sich für den Bot identifizieren kann und so den vollen Zugriff bekommt. Die Befehlssyntax wird wie folgt definiert: IDENTIFY <Bot> <Passwort>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 67: R-BS-007: IDENTIFY

4.5.5.7 Anforderungen INFO

Requirement Nr.	R-BS-008
Titel	INFO
Beschreibung	Es soll ein Befehl geschaffen werden mit dem Informationen über einen Bot angezeigt werden können. Angezeigt werden sollen:

	<p>Username, Hostname, Realname und Channels auf dem der Bot ist.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>INFO <Bot></p>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 68: R-BS-008: INFO

4.5.5.8 Anforderungen KICK

Requirement Nr.	R-BS-009
Titel	KICK
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem störende Benutzer aus dem Channel gekickt werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>KICK <Bot> <Channel> <Nickname> <Begründung></p>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 69: R-BS-009: KICK

4.5.5.9 Anforderungen LIST

Requirement Nr.	R-BS-010
Titel	LIST
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine Liste aller existierenden Bots angezeigt werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LIST</p>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 70: R-BS-010: LIST

4.5.5.10 Anforderungen KICK

Requirement Nr.	R-BS-011
Titel	MSG
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem Nachrichten mit dem Bot als Absender an den Channel gesendet werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MSG <Bot> <Channel> <Nachricht></p>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 71: R-BS-011: MSG

4.5.5.11 Anforderungen SET

Requirement Nr.	R-BS-012
Titel	SET

Beschreibung	Es soll ein Befehl geschaffen werden mit dem folgende Einstellungen für den Bot vorgenommen werden können: NAME, PASSWORD, REALNAME, USERNAME Die Befehlssyntax wird wie folgt definiert: SET <Bot> [Einstellung] <Wert>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 72: R-BS-012: SET

4.5.5.12 Anforderungen SETPASS

Requirement Nr.	R-BS-013
Titel	SETPASS
Beschreibung	Administratoren sollen das Passwort für einen Bot neu setzen können Die Befehlssyntax wird wie folgt definiert: SETPASS <Bot> <Passwort>
Kommentare	Der Benutzer muss die konfigurierten Rechte für diesen Befehl haben

Tabelle 73: R-BS-013: SETPASS

4.6 Adminserv

4.5.1 Use Case Adminserv (UC AS-01)

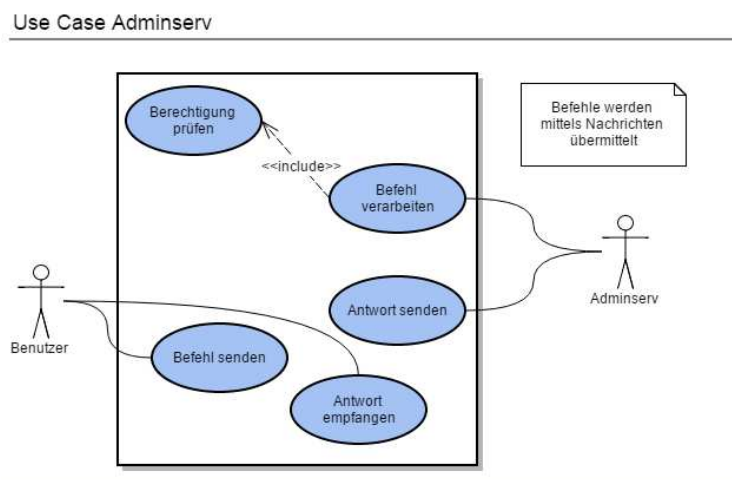


Abb. 11: Use Case Adminserv

4.6.2 Anforderungen Adminserv

Folgende Funktionen soll Adminserv anbieten

- **SAVEDATA** – Manuelles Speichern der Datenbank
- **SQUIT**– Beenden von Services

4.6.3 Anforderung Adminserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Adminserv sind identisch mit jenen von Nickserv.

4.6.4 Konfiguration Adminserv

Requirement Nr.	R-AS-001
Titel	Konfiguration Adminserv

Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Botserv anbieten:</p> <p>Abschnitt „general“ enabled: Adminserv aktivieren name: Nickname von Adminserv realname: Realname von Adminserv</p>
Kommentare	

Tabelle 74: R-AS-001: Konfiguration Adminserv

4.6.5 Anforderungen Adminserv Unterfunktionen

4.6.5.1 Anforderungen SAVEDATA

Requirement Nr.	R-AS-002
Titel	SAVEDATA
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man die gesamte Datenbankbankbank manuell speichern kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SAVEDATA</p>
Kommentare	Der Benutzer muss Services Administrator Zugriff haben.

Tabelle 75: R-AS-002: SAVEDATA

4.6.5.2 Anforderungen SQUIT

Requirement Nr.	R-AS-003
Titel	SQUIT
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man die Services aus dem Chat beenden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SQUIT</p>
Kommentare	Der Benutzer muss Services Administrator Zugriff haben.

5. Umsetzung

Nachdem wir nun alle Anforderungen dokumentiert haben müssen wir uns Gedanken über die Umsetzung machen. Dafür wollen wir uns zunächst überlegen, wie die Architektur aussehen soll. Danach überlegen wir uns, wie die Datenbank auszusehen hat und widmen uns dann der effektiven Umsetzung des Projekts.

5.1 Design

5.1.1 Architektur

Da die Dienste auf dem Basis-Server laufen und dieser wiederum eine Verbindung zum IRC Server herstellt werden die Befehle vom Client an den IRC Server gesendet. Dieser ist verantwortlich dafür, dass diese weitergeleitet werden. Da wir in den Anforderungen festgehalten haben, dass die Befehle als PRIVMSG an den jeweiligen Dienst gesendet werden müssen wir uns über die Implementierung keine Gedanken machen. Diese Art der Befehlsübermittlung bietet zudem den Vorteil, dass wir auf die Anwendung Threads verzichten können, denn der IRC-Server stellt hier eine Art „Flaschenhals“ dar da jeweils nur ein Befehl auf einmal abgearbeitet werden.

Die Daten werden jeweils beim Start in den Arbeitsspeicher geladen und sämtliche Änderungen werden im Arbeitsspeicher vorgenommen. In regelmässigen Abständen werden dann die Daten in die SQLite Datenbank geschrieben.

Das Architektur-Design sieht demnach wie folgt aus:

Architektur IRC Services

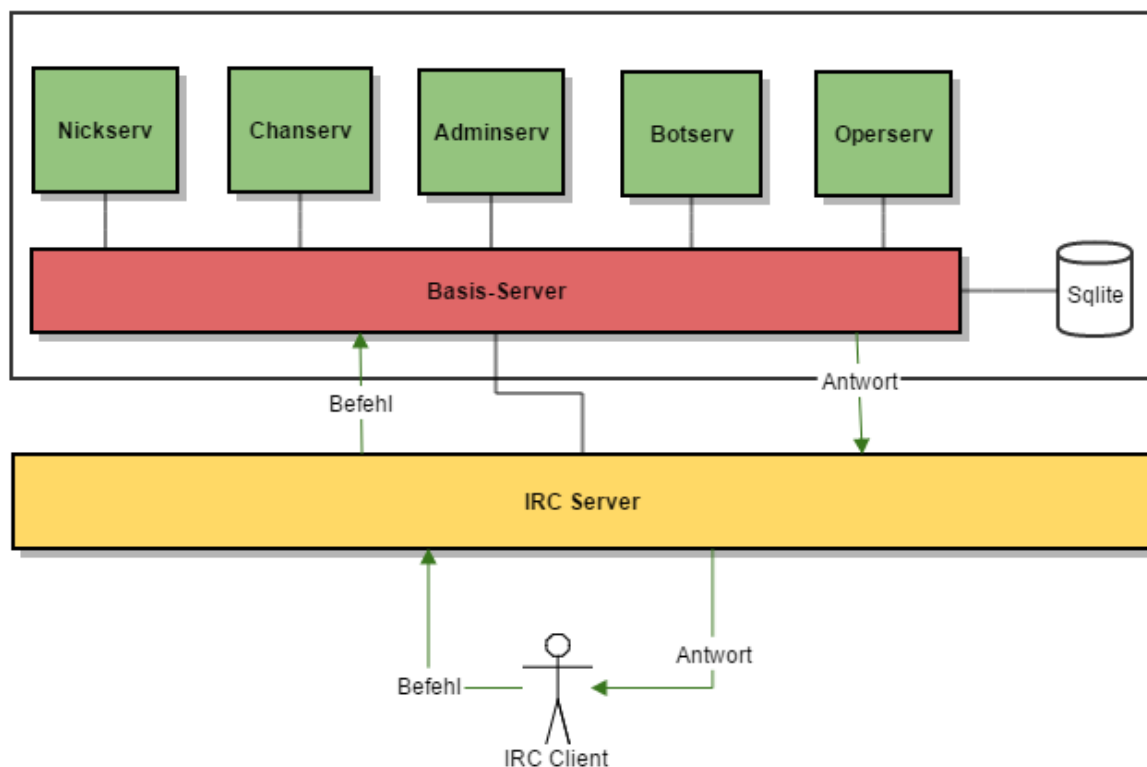


Abb. 12: Architektur

5.1.2 Datenbank-Design

Wie erwähnt werden wir als Datenbank SQLite verwenden. Das hat den Vorteil, dass die IRC Services keine weitere Software benötigt um lauffähig zu sein.

Folgendes Design wurde festgelegt:

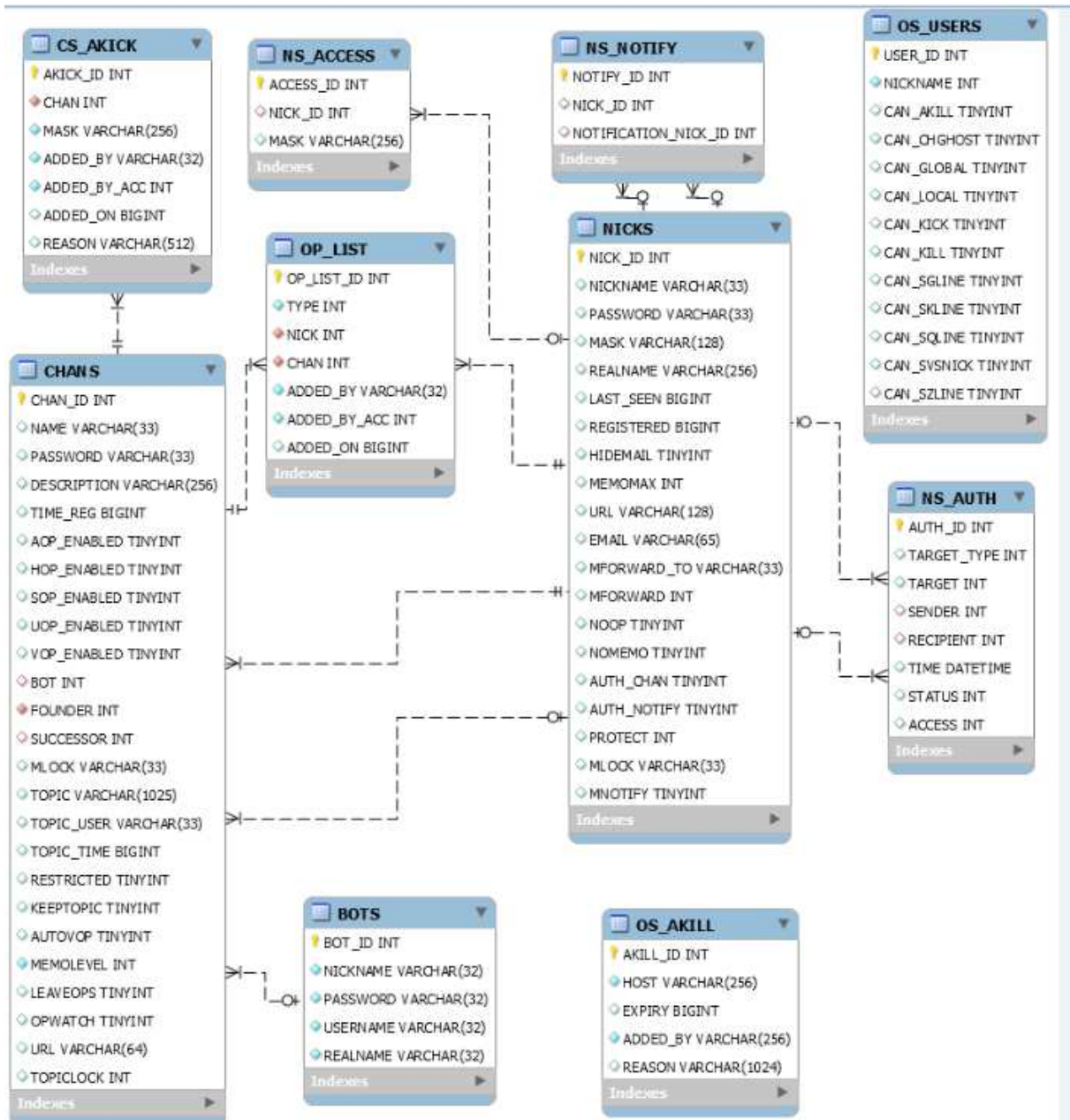


Abb. 13: Datenbank-Schema

5.2 Setup IRC Server

Der IRC Server muss zunächst so konfiguriert werden, dass er entsprechende Service-Verbindungen akzeptieren kann.

UnrealIRCd bietet als Konfiguration den Link-Block an⁵.

```
link <server-name> {
    username <usermask>;
    hostname <ipmask>;
    bind-ip <ip-to-bind-to>;
    port <port-to-connect-on>;
    password-connect <password-to-connect-with>;
    password-receive <password-to-receive> { <auth-type>; };
    hub <hub-mask>;
    leaf <leaf-mask>;
    leafdepth <depth>;
    class <class-name>;
    ciphers <ssl-ciphers>;
    options {
        <option>;
        <option>;
        ...
    };
};
```

Abb. 14: Konfiguration Unrealircd

Folgendes Beispiel stellt eine lauffähige Konfiguration dar:

```
link services.reefmaster.ch
{
    username      *;
    hostname      *;
    bind-ip       *;
    port          7029;
    hub           *;
    password-connect "pass123";
    password-receive "pass123";
    class         servers;
};
```

Abb. 15: Beispiel-Konfiguration Link

5.3. Implementierung Konfiguration

Da die Applikation für verschiedene Zwecke verwendet werden soll muss sie komplett konfigurierbar sein. Um eine Konfigurationsdatei validieren zu können gibt es verschiedene existierende Lösungen. Die Wahl fiel auf Confuse 2.5, weil die Verwendung relativ simpel ist. Confuse kommt als Bibliothek die im Quellcode eingebunden werden kann.

Die Konfigurationsdatei kann mit Confuse in Blöcke und Abschnitte unterteilt werden. Beispiel aus der Konfigurationsdatei:

```

services
{
    general
    {
        name = "services.reefmaster.ch"
        description = "Reefmaster IRC Services"
        user = "services"
        host = "reefmaster.ch"
        port = 7029
        address = "localhost"
        irc = "irc.reefmaster.ch"
    }
}

```

Abb. 16: Beispiel Basis-Server Konfiguration

Gleich nach dem Start der Applikation muss die Konfiguration geladen und validiert werden.

Als erstes wird die Konfiguration als normale Datei geöffnet. Dazu verwenden wir die `fopen` Funktion⁶:

```

int config_load(const char *file) {
    FILE *pFile;
    pFile = fopen(CONFIG_FILE, "r");
    if (!pFile) {
        printf(CONF_ERR_FILENOTFOUND "\x1b[0m");
        addlog(2, CONF_LOG_ERR_FILENOT);
        return -1;
    }
}

```

Abb. 17: config_load

Anmerkung: den Namen der Konfigurationsdatei haben wir als literarische Konstante (`CONFIG_FILE`) definiert, damit wir im Falle von Änderungen diese nur an einem Ort vornehmen müssen. Diese stellen eine Art Platzhalter dar, die durch den Prozessor beim kompilieren entsprechend ersetzt werden⁷.

Als nächstes werden die einzelnen Blöcke definiert, die validiert werden sollen. Confuse stellt eine Datenstruktur zur Verfügung in die wir einzelne Blöcke deren Abschnitte direkt bereitstellen können. Diese wird beim kompilieren verarbeitet und wir sind in der Lage, jeden Abschnitt in sich zu validieren, indem eine sogenannte Callback-Funktion definiert werden kann. Beispiel:

```

static cfg_opt_t main_general_opts[] = {
    CFG_STR_CB("name", "services.mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("user", "user", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("host", "mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_INT_CB("port", 7029, CFGF_NONE, (void*)&config_port),
    CFG_STR_CB("description", "My Net", CFGF_NONE, (void*)&config_str128),
    CFG_STR_CB("help_channel", "#help", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("address", "localhost", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("irc", "irc.mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_END()
};

```

Wir betrachten hier den Block „services“ mit dem Unterblock „general“. `cfg_opt_t` ist die Datenstruktur von Confuse. `CFG_STR_DB` ist ein Makro, also eine Art Script, die definiert wird um den Inhalt des

Makros beim kompilieren ausführt. Es verarbeitet Konfigurationsabschnitt und die angegebenen Daten in die Struktur ablegt. Folgende Parameter werden in diesem Makro erwartet:

- Name des Abschnitts
- Standard-Wert des Abschnitts, für den Fall, dass kein Wert übergeben wurde
- Spezielle Parameter (wird im gesamten Projekt nicht benötigt, daher immer CFGF_NONE)
- Zeiger auf die dazugehörige Callback-Funktion die den Abschnitt validieren soll.

Für jeden Hauptblock, wird zunächst der Unterblock mit seinen Abschnitt validiert. Danach werden alle Blöcke validiert:

```
static cfg_opt_t opts[] = {
    CFG_SEC("services", services_opts, CFGF_NONE),
    CFG_SEC("nickserv", nickserv_opts, CFGF_NONE),
    CFG_SEC("chanserv", chanserv_opts, CFGF_NONE),
    CFG_SEC("operserv", operserv_opts, CFGF_NONE),
    CFG_SEC("botserv", botserv_opts, CFGF_NONE),
    CFG_SEC("adminserv", adminserv_opts, CFGF_NONE),
    CFG_END()
};
```

Abb. 18: Validierung der Unterblöcke

Danach wird mit dem Befehl `cfg_parse` die gesamte Konfiguration geprüft und jede Callback Funktion ausgeführt. Ist etwas nicht sauber konfiguriert wird die Applikation nicht gestartet und wird stattdessen ein Fehler mit der Zeilennummer in der Konfiguration wo das Problem ist ausgegeben.

Treten keine Fehler auf, werden nun die Werte der Konfiguration den globalen Variablen, als den Variablen die in der gesamten Applikation verwendet werden, zugewiesen und die Applikation wird gestartet.

5.4. Implementierung Server-Verbindung

5.4.1 Verbindung Basis-Server

Um zu verstehen, wie ein Server sich zu einem anderen Server verbinden kann müssen wir uns zunächst über Sockets unterhalten. Ein Socket ist eine Abstraktion eines Kommunikations-Endpunktes⁸.

Ähnlich wie in einer Datei benutzen Applikationen Deskriptoren um über Socket zu kommunizieren. Um eine Netzwerk-Verbindung zu öffnen benötigen wir daher zuerst einen Socket mit seinen Einstellungen:

```
(unsigned char) dns->h_addr_list[0][3]
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
    printf(APP_ERR_SOCKET);
    return -1;
}
printf(APP_DBG_CONNECTINGTOSERVER, s unreal);
```

Abb. 19: Anlegen eines Sockets

Da das IRC Protokoll über TCP/IP Kommuniziert, verwenden wir die Option `AF_INET` (IPv4 Internet Domain). Danach müssen wir die Verbindung Konfigurieren. C stellt in der UNIX Umgebung die Struktur `sockaddr_in` zur Verfügung, die eine Adresse für einen Socket beschreiben. Damit können wir alles konfigurieren was nötig um die Verbindung herzustellen.

```
bzero(&addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons((unsigned short) port);
addr.sin_addr.s_addr = inet_addr(serverip);
```

Abb. 20: Konfiguration Socket und Adresse

Nun können wir mit `Connect` die Verbindung herstellen:

```
connect(sock, (struct sockaddr *) &addr, sizeof(addr))
```

Abb. 21: Verbindung zum Server

Wenn etwas schiefgehen sollte wird ein entsprechender Fehler ausgegeben. Steht die Verbindung muss der Basis-Server sich zum beim IRC-Server registrieren (siehe Kapitel 2.3.3). Um sich zu registrieren müssen die benötigten Nachrichten in dem vom IRC Protokoll geforderten Format an den Server senden. Um eine Nachricht zu senden verwendet man die Funktion `send`.

```
send(sock, PROT, (int) strlen(PROT), 0);
send(sock, PASS, (int) strlen(PASS), 0);
send(sock, SRV, (int) strlen(SRV), 0);
```

Abb. 22: Senden der für die Registrierung benötigten Befehle

Nun ist der Basis-Server korrekt mit dem IRC-Server verbunden. Wir kopieren die lokale Variable `sock` in die globale Variable `mainsock`. Dieser Socket ist der Socket den von der ganzen Applikation verwendet wird um Nachrichten zu senden und zu empfangen. Nun können wir die einzelnen Dienste erstellen und ebenfalls verbinden:

5.4.2 Dienste

Da ein Dienst im Sinne des Protokoll ein Nutzer mit speziellen Rechten darstellt muss sich ein solcher lediglich als normaler Benutzer zum Server verbinden. Daher wird für jeden Dienst der NICK Befehl an den Server gesendet. Im UnrealIRCd Protokoll kann man statt NICK auch „&“ senden. Wir definieren `SNICK` als Konstante für den Befehl:

```
#define SNICK "& %s 1 0 %s %s %s 0 +qdS * :%s\r\n"
```

Abb. 23: Konstante für Dienst-Verbindung

und senden den Befehl mit den benötigten Parameter an den Server:

```
extern int ns_connect(int sock) {
    char *nick = (char*) malloc(sizeof(char) * 1024);
    sprintf(nick, SNICK, ns_name, s_user,
            s_host, s_name, ns_realname);
    int rc = send(sock, nick, (int) strlen(nick), 0);
    free(nick);
    return rc;
}
```

Abb. 24: Verbindung Dienst zu Basis-Server

Nun sind die Dienste entsprechend verbunden und können Befehle verarbeiten. Damit die Verbindung offen bleibt begibt sich die Applikation nun in eine Endlosschleife:

```
/* THIS IS THE MAIN LOOP */
while(!quitting)
{
    s = recv(mainsock,buf, sizeof(ircbuf),0);
    ...
}
```

Abb. 25: Endlosschleife um die Verbindung aufrecht zu erhalten

quitting ist eine Ganzzahlvariable die beim Serverstart auf 1 gesetzt wird. Solange dieser Wert auf bleibt.

Mit recv können Nachrichten über den Socket mainsocket empfangen werden⁹.

5.4.3 Verarbeitung von Befehlen durch den Basis-Server

Damit Befehle überhaupt an den jeweiligen Dienst weitergeleitet werden können müssen diese vom Basis-Server verarbeitet und weitergeleitet werden. Wie im vorherigen Kapitel beschrieben werden Befehle mit recv über den Socket empfangen. Wir wollen nun untersuchen wie diese Befehle über den Server empfangen werden und verarbeitet werden können.

Erinnern wir uns an das Kapitel 2.3.3.5. Wir haben dort festgehalten, dass ein Befehl an einen Dienst mit PRIVMSG gesendet werden kann:

```
PRIVMSG Empfänger :Nachricht CRLF
```

Da jeder Befehl in IRC mit Carriage Return / Line Feed (CRLF) abgeschlossen werden muss können wir so einen Befehl eindeutig als solchen identifizieren. Da über den Socket die Befehl unformatiert gesendet werden muss jeweils nur den Teil bis und mit CRLF betrachtet und verarbeitet werden. Mit der C-Funktion strtok kann eine Zeichenkette nach einem gewünschten Muster abgeschnitten werden:

```
s = recv(mainsock,buf, sizeof(ircbuf),0);
if(s)
{
    buf[s] = 0;
    char *pch = strtok(buf, "\r\n");
    while(pch!=NULL)
    {
        strcpy(ircbuf,pch);
        parse();
        ircbuf[s]=0;
        pch = strtok(NULL, "\r\n");
        addlog(1,ircbuf);
    }
}
```

Abb. 26: Aufteilung der empfangenen Zeichenkette in korrekt IRC Befehle

Mit **strtok** wird nun eine korrekte IRC-Zeichenkette in die Variable **pch** abgelegt und nach **ircbuf**, die eine globale Variable ist, abgelegt. Mit der Funktion **parse()** wird diese nun verarbeitet.

Die Befehle sollen möglichst dynamisch und nicht über if-then-else Vergleiche aufgerufen werden können. Um dies zu bewerkstelligen wird die Funktion **parse()** so ausgelegt, dass der zu verarbeitende IRC Befehl in drei Teile aufgeteilt wird:

- Der Name des Befehls der aufgerufen werden soll
- Die Anzahl Argumente
- Ein Datenfeld (Array) mit den Argumenten

```

void parse(void) {
    char source[1024], cmd[1024], buf[1024], *pch, **av;
    int ac;
    irc_cmd *ic;
    strcpy(buf, ircbuf, sizeof(buf));
    if (*buf == ':') {
        pch = strpbrk(buf, " ");
        if (!pch)
            return;
        *pch = 0;
        while (isspace(*++pch))
            ;
        strcpy(source, buf + 1, sizeof(source));
        strcpy(buf, pch, sizeof(buf));
    } else {
        *source = 0;
    }
    if (!*buf)
        return;
    pch = strpbrk(buf, " ");
    if (pch) {
        *pch = 0;
        while (isspace(*++pch))
            ;
    } else
        pch = buf + strlen(buf);
    strcpy(cmd, buf, sizeof(cmd));
    ac = tokenize(pch, &av);
    if ((ic = find_cmd(cmd))) {
        if (ic->func)
            ic->func(source, ac, av);
        if (strcmp(buf, "PING") != 0)
            addlog(1, LOG_DBG_SERVERMSG, s_unreal, ircbuf);
    } else
        addlog(2, APP_ERR_UNKNOWNMSG, ircbuf);
    free(av);
}

```

Abb. 27: Die Funktion parse()

Der Inhalt des Befehls wird nun anhand der Leerzeichen weiter aufgeteilt und die erste Zeichenkette als Befehl in der Variable **cmd** abgelegt. Um die Anzahl Argumente und die Arguments selbst ablegen zu können wird die Funktion **tokenize** verwendet. Diese Funktion teilt den Rest des Befehls weiter auf und legt die einzelnen Argumente in **av** ab.

Angemerkt sei hier, dass die Variable **av** noch nicht initialisiert wurde. Dies ist hier noch nicht benötigt. Wir übergeben die Speicheradresse von **av** mittels **&av** an die Funktion **tokenize**, die nach der Aufteilung die eigentliche Ablage der Werte übernimmt:

```

int tokenize(char *buf, char ***argv) {
    int argvsize = 8;
    int argc = 0;
    char *pch;
    *argv = smalloc(sizeof(char*) * argvsize);
    while (*buf)
    {
        if(argc == argvsize)
        {
            argvsize += 8;
            *argv = srealloc(*argv, sizeof(char*) * argvsize);
        }
        if (*buf == ':')
        {
            (*argv)[argc++] = buf+1;
            buf = "";
        }
        else
        {
            pch = strpbrk(buf, " ");
            if(pch)
            {
                *pch++ = 0;
                while(isspace(*pch))
                {
                    pch++;
                }
            }
            else
            {
                pch = buf + strlen(buf);
            }
            (*argv)[argc++] = buf;
            buf = pch;
        }
    }
    return argc;
}

```

Abb. 28: Die Funktion tokenize()

In der Funktion **tokenize** sehen wir, dass hier die Variable **av** als Zeiger auf das Datenfeld definiert wurde. Dies hat den Zweck, dass nicht eine neue lokale Variable **av** benutzt werden soll, sondern diejenige, die aus **parse()** genommen werden soll. Das funktioniert, da die Variable auf dem Stack abgelegt wird. Der Aufruf von **tokenize** wird ebenfalls auf den Stack gelegt (**push**) und nach der Abarbeitung von Stack entfernt (**pop**).

Wir befinden uns nun also wieder in der Funktion **parse()**. **Tokenize** hat uns die Anzahl Argumente zurückgeliefert und die Argumente selbst in die Variable **av** abgelegt.

Nun ist also alles vorhanden, was benötigt um den Befehl auszuführen. Um nun den Befehl mit den Argumenten direkt aufzurufen benötigen wir zunächst eine Struktur, die es uns erlaubt, den Befehl dynamisch aufzurufen.


```
struct _irc_command
{
    const char *name;
    void (*func)(char *source, int ac, char **av);
};
```

Abb. 29: Struktur dynamischer IRC-Befehl

So können wir anhand des Namens die dazugehörige Funktion mit den Argumenten aufrufen. Wir erstellen eine Variable die ein Mapping zwischen einer Zeichenkette und einem Befehl beinhaltet.

```
irc_cmd irc_cmds[] = {
    { "EOS", NULL },
    { "ERROR", NULL },
    { "JOIN", c_join },
    { "KICK", c_kick },
    { "KILL", c_kill },
    { "MODE", c_mode },
    { "NETINFO", NULL },
    { "NICK", c_nick },
    { "PASS", NULL },
    { "PART", c_part },
    { "PRIVMSG", c_privmsg },
    { "PROTOCTL", NULL },
    { "PING", c_ping },
    { "QUIT", c_quit },
    { "SERVER", NULL },
    { "SMO", NULL },
    { "TOPIC", c_topic },
};

irc_cmd *find_cmd(const char *name)
{
    irc_cmd *cmd;
    for (cmd = irc_cmds; cmd->name; cmd++)
    {
        if(stricmp(name, cmd->name) == 0)
            return cmd;
    }
    return NULL;
}
```

Abb. 30: IRC-Befehle

In der Variable `irc_cmds[]` ist diese Mapping. Wir rufen also die Funktion `find_cmd` auf und geben den Namen mit. Die Variable `irc_cmds` wird durchsucht und bei einem Treffer wird der dazugehörige Befehl aufgerufen.

Wir fassen also zusammen: Wir z.B. ein `PRIVMSG` Befehl an den Server gesendet erkennt die Applikation diesen Befehl und ruft den entsprechenden Befehl im Code auf.

Wir sehen also, dass im Falle von `PRIVMSG` also die Funktion `c_privmsg` aufgerufen werden soll.

5.4.4 Benutzer- und Channelverwaltung

Damit Berechtigungen und Stati verwaltet werden können muss der Server die Benutzer und die Channels verwalten. Zu diesem Zweck werden für jeden Benutzer der sich zum Server verbindet und für jeden Channel der geöffnet wird eine entsprechende Datenstruktur angelegt, der die Attribute speichert. Diese dienen nur der Verarbeitung und werden nicht in der Datenbank gespeichert.

5.5 Implementierung Dienste

5.5.1 Allgemein

Nachdem nun der Server läuft und die Dienste verbunden sind müssen nun die Dienste implementiert werden.

Da die Dienste grundsätzlich alle eine ähnliche Implementierung aufweisen und sich nur im Inhalt unterscheiden werden wir nun die grundsätzliche Funktionsweise beschreiben.

Wie im Basis-Server sollen auch für die Dienste die einzelnen Befehle dynamisch aufgerufen werden können. Die Dazu benötigten Datenstrukturen sind identisch mit der für den Basis-Server:

```
struct _ns_cmd
{
    const char *name;
    void (*func)(char *src,int ac,char **av);
};
```

Abb. 31: Datenstruktur für Dienst-Befehle

Dazu wird für jeden Dienst eine entsprechen Such-Funktion implementiert und eine Mapping-Variable angelegt die den Befehlsnamen dem Befehl zuweist.

Jeder Befehl der an einen Dienst gesendet wird muss zunächst validiert werden. Da in der Variable ac immer die Anzahl Argumente die übergeben wird gespeichert wird zunächst die korrekte Anzahl an Argumenten überprüft. Ist diese nicht korrekt wird eine Fehlermeldung ausgegeben mit dem Hinweis auf die entsprechende Hilfsfunktion:

```
if(ac<3) {
    notice(ns_name, src, NS_RPL_REG_USAGE);
    notice(ns_name, src, NS_RPL_HLP_SHORT, ns_name, "REGISTER");
    return;
}
```

Abb. 32: Falsche Anzahl Argumente

Danach werden immer die Argumente selbst validiert. Diese unterscheiden sich nach dem Inhalt, werden aber immer gleich behandelt. Ein Argument kann eine Unterfunktion oder ein Wert sein. Kann der Dienst mit dem Argument nichts anfangen wird ebenfalls eine entsprechende Fehlermeldung ausgegeben.

```
/* check for the correct parameters */
if ((!email || !strcmp(email, "")) || (!pass || !strcmp(pass, ""))) {
    notice(ns_name, src, NS_RPL_REG_USAGE);
    notice(ns_name, src, NS_RPL_HLP_SHORT, ns_name, "REGISTER");
    return;
}
```

Abb. 33: Ungültige Argumente

5.5.2 Nickserv

Ein Nickname soll eine Entität darstellen. Jeder Nickname hat diverse Attribute, daher bietet es sich an, diese in einer Datenstruktur darzustellen:

```
struct _nickinfo {  
    NickInfo *next, *prev;  
    int id;  
    int auth_chan;  
    int auth_notify;  
    unsigned int authcount;  
    myacc *accesslist;  
    unsigned short channelcount;  
    char *email;  
    int enforced;  
    int hidemail;  
    char *last_realname;  
    time_t last_seen;  
    const char *last_usermask;  
    unsigned short memomax;  
    int mforward;  
    char *mforward_to;  
    char *mlock;  
    short mnotify;  
    char nick[NICKMAX];  
    int noop;  
    notify *notifylist;  
    int nomemo;  
    char pass[PASSMAX];  
    int protect;  
    long reserved[4];  
    time_t time_reg;  
    char *url;  
    auth *authlist;  
};
```

Abb. 34: Struktur für Nickname

Das bedeutet, dass sämtliche Information eines Nickname in dieser Datenstruktur gekapselt werden.

5.5.2.1 Registrierung Nickname

Ein Nickname wird mit dem Befehl REGISTER registriert. Nickserv ruft dazu die Funktion `ns_register` auf. Dabei werden die Eingaben validiert. Sind alle Eingaben korrekt wird eine neue Struktur angelegt. Diese Struktur wird dann in eine Liste von Strukturen (verkettete Liste) abgelegt.

Diese globale Liste wird benötigt, damit ein bestimmter gesucht werden kann und die Nicknames in der Datenbank gespeichert werden können.

Sobald der Nickname registriert wurde wird automatisch der Benutzermodus `+r` (Registrierter Nickname) gesetzt und der Benutzer erhält ein Flag, dass dieser korrekt für den Nickname identifiziert ist.

```

NickInfo *register_nick(const char *src, const char *password, char *email) {
    user *u = finduser(src);
    NickInfo *n;
    char *usermask = (char*) malloc(sizeof(char*) * 1024);
    sprintf(usermask, "%s@%s", u->username, u->hostname);
    n = calloc(sizeof(NickInfo), 1);
    if (!src)
        src = "";
    strcpy(n->nick, src, NICKMAX);
    strcpy(n->pass, password, PASSMAX);
    n->email = strdup(email);
    n->nomemo = ns_no_memo;
    n->auth_chan = ns_auth_channel;
    n->auth_notify = ns_auth_notify;
    n->protect = ns_high_protect;
    n->hidemail = ns_hide_email;
    n->noop = ns_noop;
    n->last_realname = strdup(u->realname);
    n->last_seen = time(NULL);
    n->time_reg = time(NULL);
    n->last_usermask = strdup(usermask);
    n->mforward = 0;
    if (ns_autoaccess) {
        ns_access_add_mask(n, usermask);
    }
    n->next = nicklist;
    if (nicklist)
        nicklist->prev = n;
    nicklist = n;
    return n;
}

```

Abb. 35: Ablage der Attribute in einer Struktur

5.5.2.2 Löschen eine Nickname

Soll ein Nickname gelöscht werden, wird dieser einfach aus der verketteten Liste entfernt:

```

void delete_nick(NickInfo *n) {
    if (n->prev)
        n->prev->next = n->next;
    else
        nicklist = n->next;
    if (n->next)
        n->next->prev = n->prev;
    free(n);
}

```

Abb. 36: Löschen eines Nicks

5.5.2.3 Identifikation

Wenn sich nun ein Benutzer zum Server verbindet und einen Nickname benutzt der registriert ist wird er zur Identifikation aufgefordert. Es stehen drei Schutz-Stufen zur Verfügung. Wird der Schutz auf OFF gesetzt, darf der Benutzer den Nickname benutzen, jedoch kann er natürlich keine Änderungen daran vornehmen.

Ist der Schutz auf normal gestellt, hat der Benutzer 60 Sekunden Zeit sich per Passwort zu identifizieren. Tut er das nicht, so wird der Nickname gesperrt und der Benutzer erhält einen „Guest“ Nickname mit einer Zufallsnummer

5.5.2.4 Timer

Um einen solchen Timer zu implementieren, greifen wir auf Signale zurück. Wir starten beim Programmstart den Timer indem wir ein neues Signal anlegen:

```
char buf[10000];
if(signal(SIGALRM, timer_event_handler)==SIG_ERR)
{
    printf("Error message: %s\n", strerror(errno));
    addlog(2, "Error in signal()\n");
    return;
}
```

Abb. 37: Signal

Wir entscheiden uns für den Signaltyp SIGALRM, weil mit diesem Typ das Signal beim Ablauf eines definierten Timers gesendet wird und es abgefangen werden kann¹⁰.

Wir schreiben nun eine Funktion `set_timer`, der festlegt, in welchen Abständen das Signal SIGALRM gesendet werden soll:

```
void set_timer(time_t period_in_secs) {
    struct itimerval timer_val;
    bzero(&timer_val, sizeof(timer_val));
    timer_val.it_value.tv_sec = period_in_secs;
    timer_val.it_interval.tv_sec = period_in_secs;
    if (setitimer(ITIMER_REAL, &timer_val, NULL) != 0)
        perror("Error in setitimer()");
}
```

Abb. 38: set_timer

Beim Anlegen des Signals haben wir die Handler-Funktion `timer_event_handler` angegeben. Diese wird nun in dem Abstand, den wir in der Funktion `set_timer` festgelegt haben, aufgerufen.

```
void timer_event_handler(int sigid) {
    if (sigid == SIGALRM) {
        check_timeouts();
        check_connections();
        check_expiry();
        check_akills();
        check_save();
    }
}
```

Abb. 39: timer_event_handler

In der Funktion `check_timeout` werden die Identifikationstimer, die als Datenstruktur angelegt wurden, überprüft.

5.5.2.5 Weitere Nickserv Funktionen

Für Nickserv wurden die gewünschten Funktionen implementiert. Da die detaillierte Beschreibung der Funktionen den Rahmen dieser Arbeit sprengen würde, verzichten wir an dieser Stelle darauf und verweisen auf den Quellcode.

Allgemein halten wir fest, dass für sämtliche Funktionen die benötigten Datenstrukturen als verkettete Listen angelegt und die Verarbeitung immer nach dem gleichen Muster erfolgt.

5.5.3 Chanserv

Genau wie Nickserv sollen die registrierten Channels in einer Datenstruktur festgehalten werden. Bei der Registrierung wird eine solche Struktur angelegt und in eine entsprechende verkettete Liste eingefügt.

Für jeden Befehl der über Chanserv abgesetzt wird, muss zunächst überprüft werden, ob der Benutzer überhaupt berechtigt ist, diesen Befehl zu verwenden.

5.5.3.1 Operatoren-Listen

Wie in den Anforderungen beschrieben soll Chanserv diverse Arten von Operatoren-Listen unterstützen. Um Wiederholungen im Code zu vermeiden (Programmierungs-Prinzip Don't repeat yourself, DRY) stellen wir eine Funktion zur Verfügung, die für alle Arten von Operatoren gelten soll.

```
void cs_xop_get_level(user *u, ChanInfo *c);
void cs_xop_add(char *src, char *chan, int list, char *nick);
void cs_xop_del(char *src, char *chan, int list, char *nick);
void cs_xop_list(char *src, char *chan, int list);
void cs_xop_wipe(char *src, char *chan, int list);
```

Abb. 40: Funktionsprototypen für Operatorenlisten

Die Berechtigung für den jeweiligen Befehl wird wie folgt überprüft:

```
int cs_xop_get_level(user *u, ChanInfo *c) {
    if(u->oper>cs_admin_access) {
        return ACCESS_SRA;
    }
    usernick *un = u->user_nicks;
    int level = 0;

    struct cschans *uc = u->cschans;
    while(uc) {
        if((strcmp(uc->channel, c->name) == 0) && (uc->level == CHAN_IDENTIFIED)) {
            return ACCESS_FND_FULL;
        }
        uc = uc->next;
    }
    while(un) {
        if(un->level == 2) {
            return get_access_for_nick(c, un->n);
        }
        un = un->next;
    }

    return level;
}
```

Abb. 41: Überprüfung der Berechtigung für einen Chanserv Befehl

Verfügt der Benutzer über keine Rechte, wird 0 zurückgegeben, ansonsten die Zahl, die der Berechtigung entspricht.

Sämtliche Operatoren-Berechtigungen werden in einer globalen Variablen abgelegt, um die Suche zu vereinfachen.

5.5.4 Opserv

Für Opserv ist keine komplizierte Logik erforderlich. Benutzer die noch keine Berechtigung für URC-Operatoren Befehle durch den Server haben können hinzugefügt werden und der Zugriff kann für jeden Befehl einzelnen gesetzt werden. Implementiert sind die Einzel-Berechtigung als 1 oder 0 in der entsprechenden Struktur.

5.5.4.1 AKILL

Der Autokill Befehl wird als Liste definiert. Wenn ein berechtigter Benutzer diesen Befehl ausführt wird ein Eintrag in diese Liste geschrieben. Da für diesen Befehl die Zeitdauer angegeben werden kann benötigen wir eine Implementierung, die es uns erlaubt, Kürzel wie 1w für eine Woche, oder 2y für zwei Jahre zu verwenden. Um dies zu bewerkstelligen bedienen wir uns regulären Ausdrücken:

```
#define TIME_FORMAT_H "[[:digit:]]+h"
#define TIME_FORMAT_M "[[:digit:]]+m"
#define TIME_FORMAT_W "[[:digit:]]+w"
#define TIME_FORMAT_D "[[:digit:]]+d"
#define TIME_FORMAT_Y "[[:digit:]]+y"
```

Abb. 42: Reguläre Ausdrücke für Zeitkürzel

Diese erkennen, ob eine gültige Zeitangabe definiert wurde. Mit folgendem Ausschnitt berechnen wir dann die entsprechende Zeit in Minuten:

```
}
if(match(dur, TIME_FORMAT_H)) {
    return (atol(dur) * MINUTES_PER_HOUR);
} else if(match(dur, TIME_FORMAT_D)) {
    return (atol(dur) * MINUTES_PER_DAY);
} else if(match(dur, TIME_FORMAT_W)) {
    return (atol(dur) * MINUTES_PER_WEEK);
} else if(match(dur, TIME_FORMAT_M)) {
    return (atol(dur) * MINUTES_PER_MONTH);
} else if(match(dur, TIME_FORMAT_Y)) {
    return (atol(dur) * MINUTES_PER_YEAR);
}
return 0;
```

Abb. 43: Berechnung Zeitangabe

5.5.4.2 Weitere Opserv-Befehle

Die weiteren Opserv-Befehle leiten lediglich den entsprechenden Befehl an den Server weiter der diesen dann verarbeitet.

5.5.5 Botserv

Ein Bot kann mit `bs_add` erfasst werden. Datentechnisch ist ein Bot eine Struktur mit Attributen. Die Implementierung dieses Dienstes ist relativ simpel. Sobald der Bot hinzugefügt wurde wird eine Struktur angelegt und eine Liste eingefügt. Die einzelnen Befehle sind lediglich Weiterleitungen an den IRC-Server, der die entsprechenden Befehle ausführt.

5.5.6 Adminserv

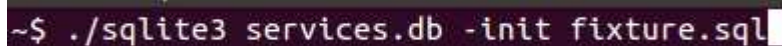
Adminserv besteht aktuell nur aus zwei Befehlen; `squit` und `savedata`. `Squit` bringt die Applikation zum stoppen und `savedata` löst eine Datenbankspeicherung aus. Details zur Datenspeicherung betrachten wir im nächsten Kapitel.

5.6 Datenbank

Wie bereits erwähnt benutzen die IRC Service eine SQLite Datenbank, da diese über eine API für C verfügt. Wir wollen auf den nächsten Seiten beschreiben wie wir mit der Datenbank umgehen wollen.

5.6.1 Erstellen Datenbank

Um alle benötigten Tabellen zu erstellen laden wir manuelle eine SQL Datei. Dazu benutzen wir das `sqlite` Tool, das wir im Projekt mitliefern.

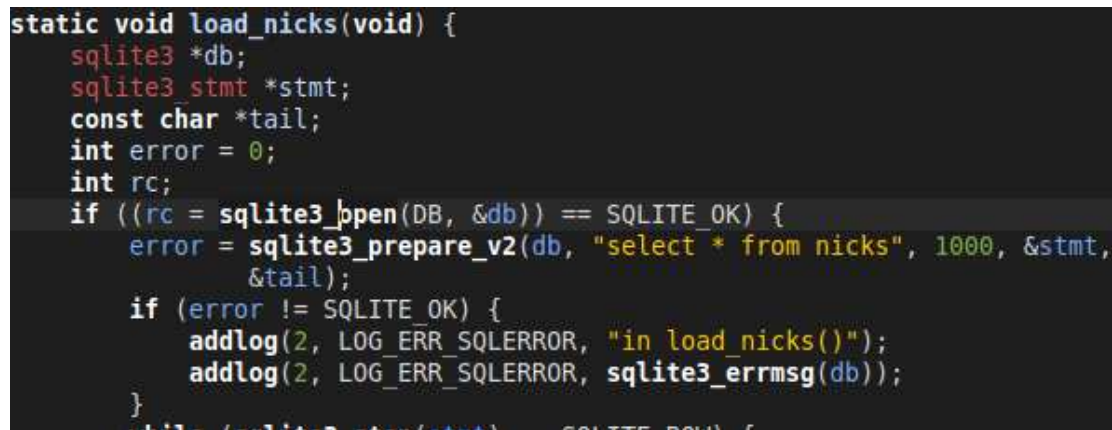


```
~$ ./sqlite3 services.db -init fixture.sql
```

Abb. 44: Erstellen der Tabellen

5.6.2 Laden der gespeicherten Daten

Beim Serverstart sollen die Daten von der Datenbank in den Arbeitsspeicher geladen werden. Dazu laden wir nacheinander die Tabellen der einzelnen Dienste. Beispielhaft wollen anhand der Nickserv Tabellen zeigen wir das funktioniert. Wir erstellen zunächst eine Datenbankverbindung:



```
static void load_nicks(void) {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    const char *tail;
    int error = 0;
    int rc;
    if ((rc = sqlite3_open(DB, &db)) == SQLITE_OK) {
        error = sqlite3_prepare_v2(db, "select * from nicks", 1000, &stmt,
                                   &tail);
        if (error != SQLITE_OK) {
            addlog(2, LOG_ERR_SQLERROR, "in load_nicks()");
            addlog(2, LOG_ERR_SQLERROR, sqlite3_errmsg(db));
        }
    }
    while (sqlite3_step(stmt) == SQLITE_ROW) {
```

Abb. 45: Vorbereiten SQLite Datenbankverbindung

Wir sehen hier, dass auch die benötigte SQL Query schon angegeben wird. Mit `sqlite3_prepare_v2` bereiten wir das benötigte Statement vor und können nun durch die einzelnen Resultate durchgehen:


```

while (sqlite3_step(stmt) == SQLITE_ROW) {
    NickInfo *n = calloc(sizeof(NickInfo), 1);
    n->id = sqlite3_column_int(stmt, 0);
    strcpy(n->nick, (char*) sqlite3_column_text(stmt, 1), NICKMAX);
    strcpy(n->pass, (char*) sqlite3_column_text(stmt, 2), PASSMAX);
    n->last usermask = strdup((char*) sqlite3_column_text(stmt, 3));
}

```

Abb. 46: Resultate

Wir legen also für jede Zeile, die von der Tabelle zurückgeliefert wird eine neue Struktur für einen Nickname und weisen die Attribute entsprechend zu.

Nachdem alle Daten geladen wurden müssen wir die Verbindung wieder schliessen:

```

}
sqlite3_close(db);

```

Abb. 47: Schliessen der Datenbankverbindung

5.6.3 Speichern der Daten

Die Daten müssen in regelmässigen Abständen gespeichert werden. Da Änderungen der Daten nur im Arbeitsspeicher und nicht direkt in der Datenbank vorgenommen werden muss immer der gesamte Datenbestand gespeichert werden. Da immer etwas schiefgehen kann verwenden wir Transaktionen, das heisst es wird entweder alles oder nicht in die Datenbank geschrieben.

```

void db_save_nicks(void) {
    sqlite3 *db;
    int query_result = 0;
    int rc;
    if ((rc = sqlite3_open(DB, &db)) != SQLITE_OK) {
        addlog(2, LOG_ERR_SQLERROR, sqlite3_errmsg(db));
        return;
    }
    sqlite3_exec(db, "BEGIN", 0, 0, 0);
    sqlite3_exec(db, "DROP TABLE IF EXISTS NICKS", 0, 0, 0);
    sqlite3_exec(db, ns_create_nicks_table, 0, 0, 0);
    NickInfo *n = nicklist;
    while (n) {
        if (!(query_result = db_add_nick(db, n))) {
            addlog(2, "Error in db_add_nick, rolling back");
            sqlite3_exec(db, "ROLLBACK", 0, 0, 0);
            sqlite3_close(db);
            return;
        }
        n = n->next;
    }
    sqlite3_exec(db, "COMMIT", 0, 0, 0);
    sqlite3_close(db);
    return;
}

```

Abb. 48: Datenbank-Transaktion

Genau wie beim Laden erstellen wir zunächst eine Datenbank-Verbindung zur Datenbankdatei. Dann starten wir die Transaktion mit BEGIN.

Nun werden alle Speicher-Queries ausgeführt. Wenn nur eine Query fehlschlägt wird mit ROLLBACK die ganze Transaktion abgebrochen und der ursprüngliche Zustand wiederhergestellt. Ansonsten werden mit COMMIT die Resultate in die Datenbank geschrieben.

5.7 Hilfe

Jeder Dienst muss eine Hilfefunktion besitzen. Mit dem HELP Befehl soll zu jedem Befehl und unterbefehl ein Hilfetext angezeigt werden können.

Das Konzept ist schnell erklärt: Wird ein Hilfetext gefunden wird dieser aus der entsprechenden Hilfe-datei gelesen und als NOTICE ausgegeben, ansonsten wird eine Fehlermeldung angezeigt. Die Hilfe-dateien sind reine Text-Dateien die von der Applikation Zeile für Zeile gelesen und ausgegeben werden.

6 Testing

6.1 Konzept

Die Installation eines Unit Test Frameworks erwies sich als äusserst mühsam und daher wurde darauf verzichtet. Stattdessen sollte jede Funktion auf jede mögliche Eingabe getestet werden.

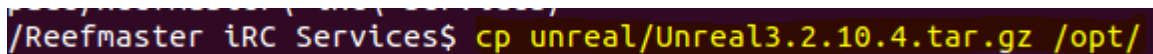
6.2 Test-Protokoll

Es wurde kein ausführliches Test-Protokoll erstellt. Jedoch wurden alle Funktionen auf alle möglichen Eingaben getestet. Bei der Entwicklung in der Programmiersprache C kommt es zudem immer wieder zu Segmentierungsfehlern. Um das Debugging zu erleichtern wurde das Tool valgrind benutzt. Mit valgrind kann die Applikation normal gestartet werden. Tritt jedoch ein Fehler auf stellt valgrind ausführliche Analyse-Resultate zur Verfügung die den Entwickler bei der Fehlersuche unterstützen.

6.3. Testing der Applikation

Um die Applikation testen zu können muss zunächst der UnrealIRCd Server installiert werden. Ein Archiv liegt diesem Projekt bei und wir wollen die Installation erläutern:

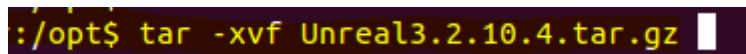
Kopieren des Archivs an einen geeigneten Ort, z.B. /opt/:



```
/Reefmaster iRC Services$ cp unreal/Unreal3.2.10.4.tar.gz /opt/
```

Abb. 49: Kopieren des Archivs

Als nächstes muss das Archiv entpackt werden:



```
:/opt$ tar -xvf Unreal3.2.10.4.tar.gz
```

Abb. 50: Entpacken des Archivs

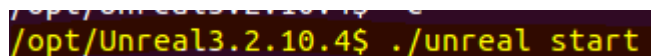
Danach wechseln wir ins das neu angelegte Verzeichnis und rufen den Befehl **Config** auf:



```
/Unreal3.2.10.4$ ./Config
```

Abb. 51: Konfiguration

Bei der Konfiguration kann alles einfach mit Enter bestätigt werden. Danach muss noch **make** ausgeführt werden um die Installation fertig zu stellen. Da UnrealIRCD konfiguriert werden muss, steht eine gültige Beispieldatei im Order „unreal“ bereit. Diese muss in das Hauptverzeichnis des Unreal-Servers kopiert werden. Danach kann der Unreal-Server wie folgt gestartet werden:



```
/opt/Unreal3.2.10.4$ ./unreal start
```

Abb. 52: Starten des UnrealIRCd

Nun läuft der Server und wir können uns den Services widmen. Im Hauptverzeichnis der Services muss nun „make“ ausgeführt werden, um die Applikation zu erstellen. Danach können wir mit ./services start die Service starten.

```
fish-guts@reefmaster:~/workspace/Reefmaster iRC Services$ ./services start
#####
#                                     #
#  Reefmaster                       #
#  IRC Services v 1.0               #
#                                     #
#####
*** Loading Configuration: services.conf...
```

Abb. 53: Starten der Services

Als IRC-Client wird der X-Chat in Linux empfohlen. Auf Ubuntu kann dieser wie folgt installiert werden:

```
$ sudo apt-get install xchat
```

Abb. 54: Installation Chat-Client

Nun kann man mit dem IRC-Client auf localhost verbinden und anfangen zu testen.

Um Befehle als Administrator zu testen wurde im UnrealIRCd ein Operatoren Benutzer vorbereitet, Benutzername und Passwort ist Admin.

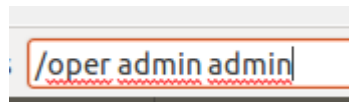


Abb. 55: Administratoren-Status erhalten

Um entsprechende Tests als IRC Operator durchzuführen wurde ein ebenfalls ein entsprechender Benutzer bereitgestellt. Benutzername und Passwort ist **ircop**

Auf weitere Hilfe wird an dieser Stelle verzichtet, da auch die Nützlichkeit der Hilfsfunktionen erprobt werden sollen.

7 Ausblick

Die Services sind schon sehr ausgebaut und können sehr viel. Dennoch sind in der Zukunft einige Modifikationen denkbar. So wird derzeit nur Englisch unterstützt. Denkbar ist eine Unterstützung für weitere Sprachen.

Adminserv unterstützt heute nur zwei Befehle. Je nach Resonanz der Benutzer kann Adminserv noch ausgebaut werden.

Die wichtigste Neuerung in zukünftigen Versionen ist bereits angedacht und in Planung: Ein Dienst zum Versenden von Kurznachrichten, Memoserv. Derzeit laufen Abklärungen, was Memoserv alles mitbringen sollte.

Da Nickserv und Chanserv bereits für diesen Dienst vorbereitet wurden dürfte sich die Komplexität der Umsetzung in Grenzen halten.

So oder so: Software lebt und darf niemals stillstehen. Anforderungen verändern sich laufend und Software muss den veränderten Anforderungen angepasst werden.

Damit entsprechendes Feedback von den Benutzern kommt soll das Projekt auf Sourceforge, einer Plattform für Open Source Software, zum Download bereitgestellt werden.

8 Fazit

Dieses Projekt war für eine Semesterarbeit sicherlich etwas gross angelegt. Da aber beim Autor dieser Arbeit ein ausgesprochenes Interesse am Thema vorhanden fiel es dennoch leicht das Projekt umzusetzen.

Technische Schwierigkeiten waren kaum vorhanden, da der Autor vom Beginn weg genaue Vorstellungen davon hatte, wie das Projekt zu implementieren sei.

Als etwas langwierig erwies sich die Erfassung der Anforderungen. Dennoch ist dies ein unverzichtbar Teil eines jeden Software-Projekt daher wurde speziell auch darauf Augenmerk gelegt.

Abschliessen möchte der Autor sich für das Interesse und den Input bedanken.

9 Anhang

9.1 Anhang A: Bilderverzeichnis

Abb. 1: Chatraum auf irc.freenode.org	9
Abb. 2: Schema eines IRC Netzwerkes	12
Abb. 3: Struktur eines IRC Netzwerkes.....	14
Abb. 4: Verbindungseinstellungen mIRC.....	16
Abb. 5: Rückgabe einer WHOIS Nachricht.....	17
Abb. 6: Use Case Basis Server.....	22
Abb. 7: Use Case Nickserv	26
Abb. 8: Use Case Chanserv	34
Abb. 9: Use Case Operserv	45
Abb. 10: Use Case Botserv.....	49
Abb. 11: Use Case Adminserv.....	54
Abb. 12: Architektur	56
Abb. 13: Datenbank-Schema.....	57
Abb. 14: Konfiguration Unrealircd.....	58
Abb. 15: Beispiel-Konfiguration Link	58
Abb. 16: Beispiel Basis-Server Konfiguration	59
Abb. 17: config_load.....	59
Abb. 18: Validierung der Unterblöcke.....	60
Abb. 19: Anlegen eines Sockets.....	60
Abb. 20: Konfiguration Socket und Adresse.....	61
Abb. 21: Verbindung zum Server.....	61
Abb. 22: Senden der für die Registrierung benötigten Befehle	61
Abb. 23: Konstante für Dienst-Verbindung.....	61
Abb. 24: Verbindung Dienst zu Basis-Server	61
Abb. 25: Endlosschleife um die Verbindung aufrecht zu erhalten.....	62
Abb. 26: Aufteilung der empfangenen Zeichenkette in korrekt IRC Befehle.....	62
Abb. 27: Die Funktion parse().....	63
Abb. 28: Die Funktion tokenize()	64
Abb. 29: Struktur dynmischer IRC-Befehl.....	65
Abb. 30: IRC-Befehle.....	65
Abb. 31: Datenstruktur für Dienst-Befehle	66
Abb. 32: Falsche Anzahl Argumente.....	66
Abb. 33: Ungültige Argumente.....	66
Abb. 34: Struktur für Nickname.....	67
Abb. 35: Ablage der Attribute in einer Struktur	68
Abb. 36: Löschen eines Nicks	68
Abb. 37: Signal.....	69
Abb. 38: set_timer.....	69
Abb. 39: timer_event_handler	69
Abb. 40: Funktionsprototypen für Operatorenlisten	70
Abb. 41: Überprüfung der Berechtigung für einen Chanserv Befehl.....	70
Abb. 42: Reguläre Ausdrücke für Zeitkürzel.....	71
Abb. 43: Berechnung Zeitangabe	71
Abb. 44: Erstellen der Tabellen	72

Abb. 45: Vorbereiten SQLite Datenbankverbindung.....	72
Abb. 46: Resultate	73
Abb. 47: Schliessen der Datenbankverbindung	73
Abb. 48: Datenbank-Transaktion	73
Abb. 49: Kopieren des Archivs.....	75
Abb. 50: Entpacken des Archivs	75
Abb. 51: Konfiguration	75
Abb. 52: Starten des UnrealIRCd	75
Abb. 53: Starten der Services	76
Abb. 54: Installation Chat-Client	76
Abb. 55: Administratoren-Status erhalten	76

9.2 Anhang B: Tabellenverzeichnis

Tabelle 1: Stakeholder.....	22
Tabelle 2: R-S-001: Unterstützte IRC Server.....	23
Tabelle 4: R-S-002: Konfiguration Basis-Server	23
Tabelle 5: R-S-003: Starten des Servers.....	23
Tabelle 6: R-S-004: Validierung Server-Konfiguration.....	24
Tabelle 7: R-S-005: Daten von Datenbank laden.....	24
Tabelle 8: R-S-006: Verbindung zum IRC Server.....	24
Tabelle 9: R-S-007: Dienste starten und Verbinden.....	24
Tabelle 10: R-S-008: Daten in regelmässigen Abständen speichern	25
Tabelle 11: Stakeholder Nickserv	26
Tabelle 12: Akteure Nickserv.....	26
Tabelle 13: R-NS-001: Nickserv Hauptfunktion	27
Tabelle 14: R-NS-002: Konfiguration Nickserv	28
Tabelle 15: R-NS-003: ACC.....	29
Tabelle 16: R-NS-004: ACCESS	29
Tabelle 17: R-NS-005: AUTH	30
Tabelle 18: R-NS-006: DROP	30
Tabelle 19: R-NS-007: GETPASS.....	30
Tabelle 20: R-NS-008: GHOST	31
Tabelle 21: R-NS-009: IDENTIFY	31
Tabelle 22: R-NS-010: INFO	31
Tabelle 23: R-NS-011: LIST.....	31
Tabelle 24: R-NS-012: LISTCHANS	32
Tabelle 25: R-NS-013: NOTIFY	32
Tabelle 26: R-NS-014: REGISTER.....	32
Tabelle 27: R-NS-015: RELEASE	33
Tabelle 28: R-NS-016: SET	33
Tabelle 29: R-NS-016: SETPASS	33
Tabelle 30: R-NS-017: Identifikationstimer	33
Tabelle 31: R-CS-002: Konfiguration Chanserv.....	37
Tabelle 32: R-CS-002: ACC.....	37
Tabelle 33: R-CS-003: AKICK.....	38
Tabelle 34: R-CS-004: AOP.....	38
Tabelle 35: R-CS-005: DE-/HALFOP	38
Tabelle 36: R-CS-006: DE-/OP.....	39

Tabelle 37: R-CS-007: DEVOICE	39
Tabelle 38: R-CS-008: DROP	39
Tabelle 39: R-CS-009: GETPASS	40
Tabelle 40: R-CS-010: HOP	40
Tabelle 41: R-CS-011: IDENTIFY	40
Tabelle 42: R-CS-013: INVITE	41
Tabelle 43: R-CS-013: LIST	41
Tabelle 44: R-CS-014: MDEOP	41
Tabelle 45: R-CS-015: MKICK	41
Tabelle 46: R-CS-016: REGISTER	42
Tabelle 47: R-CS-017: SET	42
Tabelle 48: R-CS-018: SETPASS	43
Tabelle 49: R-CS-019: SOP	43
Tabelle 50: R-CS-020: UNBAN	43
Tabelle 51: R-CS-021: UOP	44
Tabelle 52: R-CS-022: Uop	44
Tabelle 53: R-OS-001: Konfiguration Opserv	46
Tabelle 54: R-OS-002: AKILL	46
Tabelle 55: R-OS-003: CHATOPS	47
Tabelle 56: R-OS-004: CHGHOST	47
Tabelle 57: R-OS-005: GLOBAL	47
Tabelle 58: R-OS-006: KILL	48
Tabelle 59: R-OS-007: LOCAL	48
Tabelle 60: R-OS-008: OPER	48
Tabelle 61: R-OS-009: SGLINE / SKLINE / SZLINE / SQLINE	49
Tabelle 62: R-OS-001: Konfiguration Botserv	51
Tabelle 63: R-BS-002: ADD/DEL	51
Tabelle 64: R-BS-003: DE-/HALFOP	51
Tabelle 65: R-BS-004: DE-/VOICE	52
Tabelle 66: R-BS-005: DE-/OP	52
Tabelle 67: R-BS-006: GETPASS	52
Tabelle 68: R-BS-007: IDENTIFY	52
Tabelle 69: R-BS-008: INFO	53
Tabelle 70: R-BS-009: KICK	53
Tabelle 71: R-BS-010: LIST	53
Tabelle 72: R-BS-011: MSG	53
Tabelle 73: R-BS-012: SET	54
Tabelle 74: R-BS-013: SETPASS	54
Tabelle 75: R-AS-001: Konfiguration Adminserv	55
Tabelle 76: R-AS-002: SAVEDATA	55

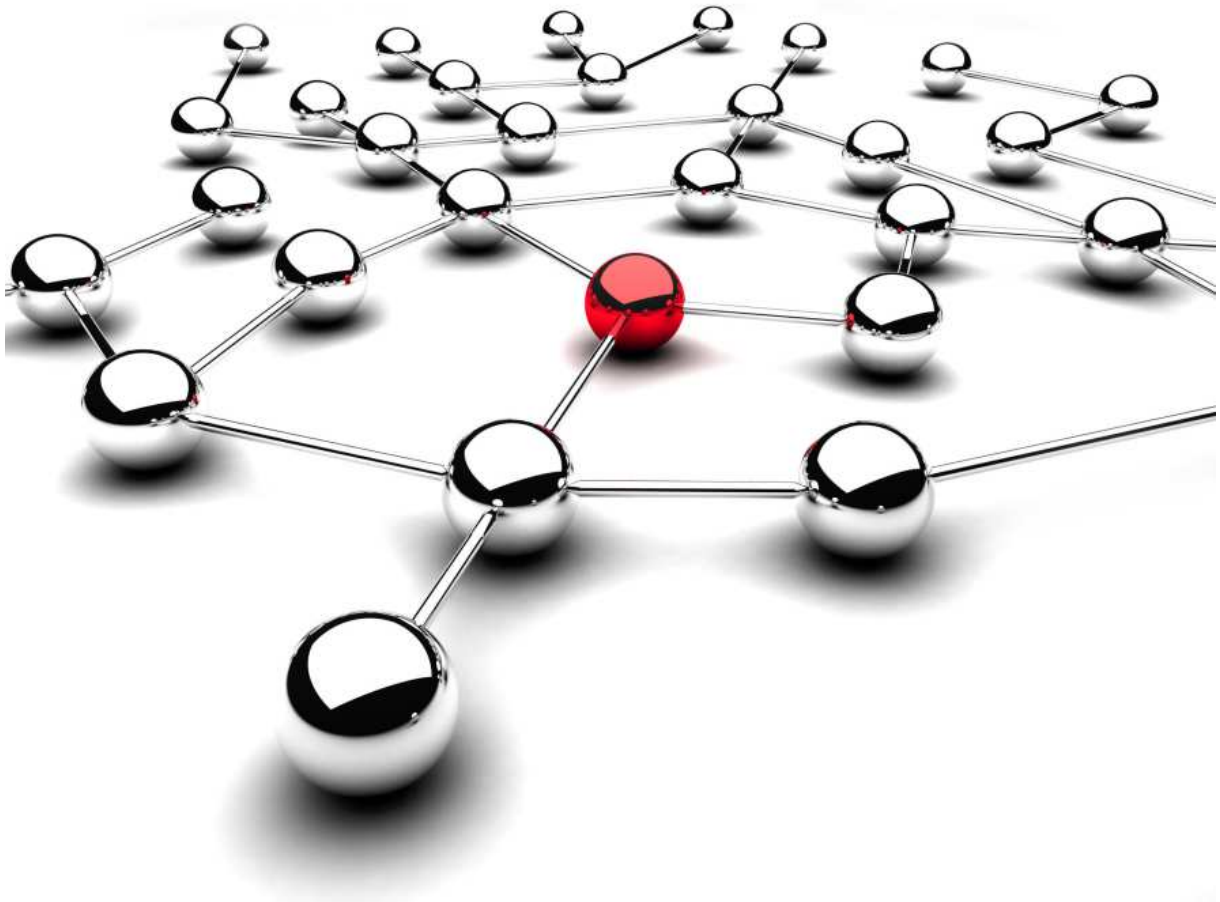
9.4. Literaturverzeichnis

- [1] http://www.selflinux.org/selflinux/html/irc_geschichte01.html abgerufen 01.01.2015
- [2] https://de.wikipedia.org/wiki/Internet_Relay_Chat#Entwicklung, abgerufen 01.01.2015
- [3] <http://www.at-mix.de/internet/internet-0207.htm>, abgerufen am 01.01.2015
- [4] <http://www.webmaster.com/crtabeditions.htm>, abgerufen am 02.01.2015
- [5] <https://www.unrealircd.org/files/docs/unreal32docs.html#ulinesblock>, abgerufen am 21.02.2014

- [6] Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language. Englewood Cliffs, Vereinigte Staaten: Prentice Hall, 50. Auflage, 2012, Seite 177. ISBN 0-13-119362-8
- [7] Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language. Englewood Cliffs, Vereinigte Staaten: Prentice Hall, 50. Auflage, 2012, Seite 14. ISBN 0-13-119362-8
- [8] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 590. ISBN 978-0-321-63773-4
- [9] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 612. ISBN 978-0-321-63773-4
- [10] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 317. ISBN 978-0-321-63773-4
- Titelbild: <http://www.sankt-mauritz.com/sites/default/files/bilder/nachricht/netzwerk.jpg>, abgerufen am 29.10.2014s

IRC Services in C

Diensteserver für einen Internet-Chat



Zürcher Hochschule für angewandte Wissenschaften
Betreuungsperson: Karl Brodowsky, Fachdozent für Systemprogrammierung
Erscheinungsjahr: 2015

Eine Semesterarbeit von Severin A. Müller

Abstract

Die vorliegende Arbeit behandelt die Entwicklung eines Dienste-Server für einen Text-Chat basierend auf dem Internet-Relay Chat (IRC) Protokoll. Die Einleitung schildert die Ausgangslage, die Motivation des Autors und die Ziele dieser Arbeit. Im zweiten Kapitel folgt ein Exkurs über die Geschichte des Internet Relay Chat und die Einführung in dessen Funktionsweise und das Protokoll. Darauf folgt in einer Marktanalyse eine Übersicht bestehender Lösungen sowohl für den reinen Chat als auch für Dienste oder gar integrierte Lösungen. Im vierten Kapitel folgt eine ausführliche Anforderungsanalyse in welcher jede Funktion die das zu entwickelnde Programm unterstützen soll detailliert beschrieben wird. Dazu wurde jeweils ein entsprechender Anwendungsfall erstellt und die Architektur in einer Grafik anschaulich dargestellt. Im darauffolgenden Kapitel wird auf die Umsetzung eingegangen. Da viele Code-Abschnitte sich wiederholen wurde jeweils das erwähnt, was für diese Arbeit von Interesse ist. Speziell auf die Aspekte der Systemprogrammierung (Signale, Timer) wird ein Augenmerk gelegt. Zudem bietet das einen Überblick über die Anwendung der Datenbank SQLite. Im Kapitel über das Testing findet sich ein kurzes Testing-Konzept sowie eine Anleitung wie das Programm korrekt aufgesetzt und betrieben werden. Das entsprechende Benutzerhandbuch ist Teil des Programms und wird in einem Abschnitt kurz beschrieben. Die Arbeit schliesst mit einem Ausblick auf zukünftige Funktionen und ein Fazit.

Der Autor konnte eine funktionierende Lösung in der vorgegebenen Zeit entwerfen und implementieren. Die Arbeit war sicherlich umfangreich, aber dank der Motivation des Autors wurden die Ziele dieser Arbeit erreicht.

Inhaltsverzeichnis

1 Einleitung	9
1.1 Themenwahl	9
1.2 Ziele der Arbeit.....	10
1.3 Aufgabenstellung	11
2 Einführung in IRC und Services	12
2.1 Geschichte des IRC.....	12
2.2 Funktionsprinzip des IRC.....	12
2.3 Protokoll	13
2.3.1 Einführung.....	13
2.3.1.1 Grundlagen	13
2.3.1.2 Operatoren	14
2.3.1.3 Channels	14
2.3.2 Die IRC Spezifikation	14
2.3.3 IRC Konzepte	14
2.3.3.1 1:1 Kommunikation	15
2.3.3.2 1:n Kommunikation	15
2.3.3.3 1:Alle Kommunikation	15
2.3.3 Nachrichten Details.....	15
2.3.3.1 Pass Nachricht	15
2.3.3.2 Nick Nachricht	15
2.3.3.3 User Nachricht.....	16
2.3.3.4 Server Nachricht.....	17
2.3.3.5 PRIVMSG.....	17
2.3.3.6 NOTICE.....	17
2.4 Services	17
2.4.1 Einführung.....	17
2.4.2 Nickserv	18
2.4.3 Chanserv.....	18
2.4.4 Opserv.....	18
2.4.5 Botserv	18
2.4.6 Adminserv	19
2.4.7 Festlegung der Grenzen	19

3 Marktanalyse	20
3.1 Webmaster ConferenceRoom.....	20
3.2 Unreal IRCd	20
3.3 IRC Services	20
3.4. Anope Services	20
3.5 Auspice.....	21
3.6 Fazit.....	21
4 Anforderungen	22
4.1 Basis-Server	22
4.1.1 Use Case Basis Server (UC S-01).....	22
4.1.2 Anforderungen Basis-Server	22
4.1.2.1 IRC Server	22
4.1.2.2 Konfiguration Basis-Server	23
4.1.2.3 Starten und Stoppen des Servers	23
4.1.2.4 Validierung Server-Konfiguration.....	23
4.1.2.5 Daten von Datenbank laden.....	24
4.1.2.6 Verbindung zum IRC Server.....	24
4.1.2.7 Dienste starten und verbinden.....	24
4.1.2.8 Daten in regelmässigen Abständen speichern	24
4.1.2 Dienste allgemein	25
4.1.3 Datenbank.....	25
4.1.4 Logging	25
4.2 Nickserv.....	26
4.2.1 Use Case Nickserv (UC NS-01).....	26
4.2.2 Anforderungen Nickserv	26
4.2.3 Anforderung Nickserv Hauptfunktion.....	27
4.2.4 Konfiguration Nickserv.....	27
4.2.5 Anforderungen Nickserv Unterfunktionen	29
4.2.5.1 Anforderungen ACC.....	29
4.2.5.2 Anforderungen ACCESS	29
4.2.5.3 Anforderungen AUTH	30
4.2.5.4 Anforderungen DROP	30
4.2.5.4 Anforderungen GETPASS.....	30

4.2.5.5 Anforderungen GHOST	30
4.2.5.6 Anforderungen IDENTIFY.....	31
4.2.5.7 Anforderungen INFO	31
4.2.5.8 Anforderungen LIST	31
4.2.5.9 Anforderungen LISTCHANS.....	31
4.2.5.10 Anforderungen NOTIFY	32
4.2.5.11 Anforderungen REGISTER.....	32
4.2.5.12 Anforderungen RELEASE	32
4.2.5.13 Anforderungen SET.....	33
4.2.5.14 Anforderungen SETPASS.....	33
4.2.6 Weitere Nickserv Anforderungen	33
4.2.6.1 Identifikationstimer	33
4.3 Chanserv	34
4.3.1 Use Case Chanserv (UC CS-01).....	34
4.3.2 Anforderungen Chanserv	34
4.3.3 Anforderung Chanserv Hauptfunktion	35
4.3.4 Konfiguration Chanserv	35
4.3.5 Anforderungen Chanserv Unterfunktionen.....	37
4.3.5.1 ACC	37
4.3.5.2 Anforderungen AKICK.....	37
4.3.5.3 Anforderungen AOP	38
4.3.5.4 Anforderungen DE-/HALFOP	38
4.3.5.5 Anforderungen DE-/OP	39
4.3.5.6 Anforderungen DE-/VOICE	39
4.3.5.7 Anforderungen DROP	39
4.3.5.8 Anforderungen GETPASS.....	39
4.3.5.9 Anforderungen HOP	40
4.3.5.10 Anforderungen IDENTIFY.....	40
4.3.5.11 Anforderungen INVITE.....	40
4.3.5.12 Anforderungen LIST	41
4.3.5.13 Anforderungen MDEOP.....	41
4.3.5.14 Anforderungen MKICK.....	41
4.3.5.15 Anforderungen REGISTER.....	42
4.3.5.16 Anforderungen SET.....	42
4.3.5.17 Anforderungen SETPASS.....	43
4.3.5.18 Anforderungen SOP.....	43

4.3.5.19 Anforderungen UNBAN	43
4.3.5.20 Anforderungen UOP	44
4.3.5.21 Anforderungen VOP	44
4.4 Operserv.....	45
4.4.1 Use Case Operserv (UC OS-01)	45
4.4.2 Anforderungen Operserv	45
4.4.3 Anforderung Operserv Hauptfunktion.....	45
4.4.4 Konfiguration Operserv.....	45
4.4.5 Anforderungen Operserv Unterfunktionen	46
4.4.5.1 Anforderungen AKILL.....	46
4.4.5.2 Anforderungen CHATOPS.....	47
4.4.5.3 Anforderungen CHGOST.....	47
4.4.5.4 Anforderungen GLOBAL	47
4.4.5.5 Anforderungen KILL.....	47
4.4.5.6 Anforderungen LOCAL.....	48
4.4.5.7 Anforderungen OPER	48
4.4.5.8 Anforderungen Server-Bans und Nicksperrern	49
4.5 Botserv	49
4.5.1 Use Case Botserv (UC BS-01)	49
4.5.2 Anforderungen Botserv	50
4.5.3 Anforderung Botserv Hauptfunktion	50
4.5.4 Konfiguration Botserv	50
4.5.5 Anforderungen Botserv Unterfunktionen	51
4.5.5.1 Anforderungen ADD/DEL	51
4.5.5.2 Anforderungen DE-/HALFOP	51
4.5.5.3 Anforderungen DE-/VOICE	51
4.5.5.4 Anforderungen DE-/OP	52
4.5.5.5 Anforderungen GETPASS.....	52
4.5.5.6 Anforderungen IDENTIFY.....	52
4.5.5.7 Anforderungen INFO	52
4.5.5.8 Anforderungen KICK	53
4.5.5.9 Anforderungen LIST.....	53
4.5.5.10 Anforderungen KICK	53
4.5.5.11 Anforderungen SET.....	53
4.5.5.12 Anforderungen SETPASS.....	54

4.6 Adminserv	54
4.5.1 Use Case Adminserv (UC AS-01)	54
4.6.2 Anforderungen Adminserv.....	54
4.6.3 Anforderung Adminserv Hauptfunktion	54
4.6.4 Konfiguration Adminserv	54
4.6.5 Anforderungen Adminserv Unterfunktionen	55
4.6.5.1 Anforderungen SAVEDATA.....	55
4.6.5.2 Anforderungen SQUIT	55
5. Umsetzung	56
5.1 Design	56
5.1.1 Architektur	56
5.1.2 Datenbank-Design.....	57
5.2 Setup IRC Server	58
5.3. Implementierung Konfiguration	58
5.4. Implementierung Server-Verbindung	60
5.4.1 Verbindung Basis-Server	60
5.4.2 Dienste	61
5.4.3 Verarbeitung von Befehlen durch den Basis-Server.....	62
5.4.4 Benutzer- und Channelverwaltung	66
5.5 Implementierung Dienste	66
5.5.1 Allgemein	66
5.5.2 Nickserv	67
5.5.2.1 Registrierung Nickname	67
5.5.2.2 Löschen eine Nickname.....	68
5.5.2.3 Identifikation	68
5.5.2.4 Timer	69
5.5.2.5 Weitere Nickserv Funktionen.....	70
5.5.3 Chanserv.....	70
5.5.3.1 Operatoren-Listen	70
5.5.4 Opserv.....	71
5.5.4.1 AKILL	71
5.5.4.2 Weitere Opserv-Befehle	71
5.5.5 Botserv	72

5.5.6 Adminserv	72
5.6 Datenbank.....	72
5.6.1 Erstellen Datenbank.....	72
5.6.2 Laden der gespeicherten Daten.....	72
5.6.3 Speichern der Daten	73
5.7 Hilfe.....	74
6 Testing	75
6.1 Konzept	75
6.2 Test-Protokoll.....	75
6.3. Testing der Applikation	75
7 Ausblick.....	77
8 Fazit	78
9 Anhang.....	79
9.1 Anhang A: Bilderverzeichnis	79
9.2 Anhang B: Tabellenverzeichnis	80
9.4. Literaturverzeichnis.....	81

1 Einleitung

1.1 Themenwahl

Wir leben im Zeitalter von Web 2.0 wodurch der klassische Text-Chat an Bedeutung verloren hat. Schliesslich sind Web-Browser heute fähig Videos abzuspielen und Webcam Übertragungen durchzuführen. IRC als reiner Text-Chat spielt also nur noch eine untergeordnete Rolle bei den Anwendern.

Dennoch ist der IRC (Internet Relay Chat) in UNIX-Kreisen immer noch äusserst populär. Nehmen wir zum Beispiel den Chat auf irc.freenode.org. Dort finden sich auch heute noch zehntausende Benutzer die gleichzeitig online sind. Sie treffen sich themenbasierten Räumen wie zum Beispiel `#linux` oder `#C` und bilden so eine riesige Gemeinschaft um bei Fragen zu den jeweiligen Themen zu helfen.

```

[01:24:27] * Now talking in #c!
[01:24:27] * Topic is "The C Programming Language || C11 is the current C Standard || PASTE (>3 lines) here: http://ideone.com/ || HOME page: http://www.iso-9899.info/ (to edit you need an account, see a wiki/channel admin) || BOOKS here: http://www.iso-9899.info/wiki/Books || Be civil"
[01:24:27] * Set by pragma- on Mon Dec 22 18:58:57 2014
[01:24:27] * ##c Fish-Guts mikedugan r3g3x [spoiler] robot-beethoven doppel dm7freak Muzer neurodrone cousteau netj desantis arescorpio entitz schleppel Jubb Love4Boobies phinx
Albori asakura frostschutz bolt Uffalizer kyllerisse1010 dardevlin mgorbach Trondby matp jeffieismyhero nistyre_ moop aslant immbis Saporok_ anonnumberanon Lowl3v3 plitter
zymurgy Pessimist oleo Tordek roentgen likecolacola BlueShark_ KindOne Enthralled vishwin n1000_ neuron moparstbest mulvane
[01:24:27] * ##c Mr_Sheesh Beethy wrl Photon12 KWhat4 DarthTetris Wamanuz Anaphaxeton raj TMM screwjack GinoMan delta-nry tigger0jk x1 dts[pokeball Jesin fengshaun Ephexev
FreezingCold rrm_ pnbearst shikhn MrCoffee cathaur NeoGeo64 flaviu junaidnaseer2 bxomonjhome tharugrim killertester Celelbi freeruncan Noldorin hat] raboof cpt_yossarian
hiptoeubic kesselhaus bone Khisanth BreadProduct Nothing4You Atrumx Haswell Biohazard BoredHamster Zentdayn heurist Suchorski Blaguvest
[01:24:27] * ##c Orion] sparetire_ neionz sjohnson tapout brenns10 jamiec fstd prc1 bluss DarkFazy M-ou-se krokus apeiros_ d3m1g0d-Left_Turn Synchunk fizzie neo1691 garner jzk
sharky ivan] day cebor_ Cabanossi Riviera Enissay greenbagels megaTherion araujo ChoiKySang turnedtodust whaletchno sivoais NIN101 nicedice ltd yarker Vutral Exagone313 Auv
Irrelum CHC heftig_ dexter0 KrzaQ Farow Jaood Norrin cjwelborn msy cysm Gravitrion slxpk rcombs jz] george2 Polarina bluehavana
[01:24:27] * ##c Bigcheese synapt notion Nidkeeh deste a-a skarn gbarboza ConstantineXVI viruser Atlantic777 CrypticSquared jgeboski drdanick cats toxboi_ pippijn xace kst notpara
mathewji terabit flounders sheilong Anoniem4 ss mnb_ CJKay BSaboia_ liberal l0rdn1x quinman22 kalzz bb010g Karethoth Fenhl doudcell hellschreiber superjudge caen23 ekneuss
starship Zerok nuopogdi Talsman twoface88 stuxjRC-only dorp jds camerin tsuyoi homebrew rud93 bpmdeley R0b0t1 antoxyz lens
[01:24:27] * ##c Ard1t cpt-oblivious thomas CMGauger tempus_ fol kermi Nik05 lyindArk robbyconnor dusted Vohveli Innominate glebofff Natch ncjb widmo pHcF mervaka keel vicenteH
davor edr ElectricSheep DarthGandalf asmx marvimias DJZOO! SM0TVI orbitz ark_ de-vri-es cyberspace- SiegeX AsadH drakored lytchi scrapcode ormaaj Lokomotiv james41382 Trivium
polygraphed BGL roboman2444 dx boru ishkawa m0shbear barfod [Awaxx] Spark BombStrike glukt Zmsky jayme Conquer1Warrior MrC
[01:24:27] * ##c Praise _ether_ SirCnpwin dozn Alina-malina Jagged rol31 w41 Steffanx iron_houzi kirin_ yayachiken Ulrike_Rayne kurti wapifapi tazle @candide iveqy bool_ ironode0
proZaC Inode Burgundy jriese j_wright Niamik riotz sabotender KnownSyntax toothrot Saban sabalaba nitrix pestle benwaffle byaruhaf bdamos gf3 sono Brando753 freanux sbahra
stonecolddevin dagle myte: Defaultt izabera xpog tormoz KMA gojdfish zid_ Kostenko_ introom froggyman pragma- lanhedoesit
[01:24:27] * ##c kniskropd freakazoid0223 dami Cavedude Number 2 klarrt BSDBlack kate eljraz notker RichiH letoram shibumi diethyl fishes 1 robink nukeddx blackbit alip ashkan kbt
DarkCthulhu kiddo Tabmow jrib Trieste Thermi Taylor genpaku BlastHardcheese Trashlord paracyst rob_ _ refx_ cherwin spoty staykov renopt hohum jv Plazma Platini talanor dh
Robdgreat Aero Vorpal croz XgF phantomcircuit huntercool relaxed Xgc RTG_ Triskelos lykkn mthowe altf2o tm512 pumba MindlessDrone
[01:24:27] * ##c Mellett68 Shayanjm yokel tomaw tajnyman ccaione cross rascal999 catern fold Chris tsstc andres Kasreyn dostoyevsky AndrewX192 geidi rvncerr impulse150 marky LoRez
Happzz tobiasBora Puculowski debris_ thomney monokrome ra4king dualbus HylianSavior haasn fatpudding pandzilia LiamM gl barometz Fill kline Buzzer tomodachi rigid goncalo mofino
Greatgib icedev Hearbroken sparetire CaZe baggio Cerise flam_ kloeri heinrich5991 caveat- dav1d impulse emias mountaingoat
[01:24:27] * ##c Nagra shvouchk awake iptable Bane^_ hennilu DarkGhost arch_ dutc koolman jiffe beneth_ keltvek chewbranca leprechau rolst SporkWitch bitdigger pa Cleo kov
salamanderrake chishiki csuiu boos intoX27h pstef ski bumber_ Yaknot5 f0lder LeoNerd ohama richardwhiuk oblique Cyclohexane octanium urlgrabber Virox K1773R S_T_A_N jon_y
epsylon hrnz monty metalys SHODAN octe b00k3r Affliction Gentle codebrainz Paradox924k bps Zarthus ribasushi Evil3Stoker AStorm Tutkah
[01:24:27] * ##c dtsf anunnaki Draconx Stary2001 shiftplusone mischief machty_ Jupelius kotog dxtx mit averell KingOfKarlsruhe edk aandrew bablen wio7 gagan662 APic unreal Zhivago
Thanat0s_cnu- MatheusOI sharpneli dv- wdouglas nutron Fuux yorick kern edwardly twkm seg ivan_ stfn kAworu sazzter timemage divine Ewomire FergusL elwinto Oddity petern_
[01:24:27] * Users: [ @Ops:1 +Voiced:0 Regular:555 Total:556 ]
[01:24:27] * Modes: +Cnrtf #c-unregistered
[01:24:27] * 1164523359
[01:24:28] * http://www.iso-9899.info/
[01:24:31] <doppel> which means i'll be around to bug you for years to come. how lucky are you.

```

Abb. 1: Chatraum auf irc.freenode.org

Auch wenn die Benutzer heute auf andere Chat-Technologien ausweichen heisst das nicht, dass der IRC am Aussterben ist. Viele Benutzer in der Linux-Welt machen sich nichts aus grafisch hochstehenden Flash-Chats sondern möchten schnell und einfach Informationen austauschen.

Da jedoch nicht immer alle Benutzer freundliche Absichten haben und sich teilweise daneben benehmen benötigt man Mittel um sich solch ungebeter Gäste zu entledigen. Dazu dienen sogenannte Operatoren in Räumen. Sie haben Rechte, die „normale“ Benutzer nicht haben. So können Operatoren zum Beispiel einen Benutzer „kicken“, also aus dem Raum werfen oder „bannen“ also verbannen, sodass dieser Benutzer den Raum nicht mehr betreten kann.

Das Protokoll auf dem IRC zugrunde liegt sieht vor, dass der erste Benutzer der den Raum betritt automatisch den Operatoren-Status erhält. Er kann weitere Operatoren bestimmen und hat weitere Rechte.

Dadurch entsteht jedoch das Problem, dass wenn nur ein Operator im Raum ist und er diesen verlässt niemand mehr Operatoren-Rechte hat, bis alle weiteren Benutzer den Raum verlassen und dieser so neu erstellt werden kann.

Ein weiteres Problem ist, dass das Protokoll keine Speicherung von Benutzernamen (sog. „Nickname“) zulässt. Wenn also ein Benutzer den gewünschten Namen bereits innehat gibt es keine Möglichkeit denselben Namen zu verwenden.

Um diesen und weiteren Problemen zu begegnen wurden sogenannte „Services“ erschaffen. Diese bieten zum Beispiel die Möglichkeit, Nicknames oder Chaträume (sog. Channels) zu registrieren und so vor unbefugtem Zugriff zu schützen.

In einigen bestehenden Lösungen sind diese bereits enthalten, jedoch gibt es zahlreiche Server, die ohne Services ausgeliefert werden.

Der Autor der vorliegenden Arbeit war lange Zeit als Moderator auf dem Chat von Swisscom (damals Bluewin) tätig. Dieser Chat lief auf einer proprietären Lösung mit integrierten Services. Nach der Abschaltung des Chat begannen einige Nutzer eigene Chats aufzusetzen, dies mit Open-Source Lösungen.

Da viele Open-Source Lösungen keine integrierten Services haben, stellte sich so bald die Frage nach einer entsprechenden Lösung. Es gibt einige gute und populäre Services-Lösungen, jedoch unterscheiden diese sich stark und viele Benutzer wünschten sich, dass die Befehle ähnlich wie auf dem Bluewin Chat aufgebaut seien. Um dieses Bedürfnis zu befriedigen entschloss sich der Autor eine entsprechende Lösung zu entwickeln. Die vorliegende Arbeit befasst sich mit dieser Lösung.

1.2 Ziele der Arbeit

Um die im Kapitel 1.1 beschriebenen Probleme zu lösen und die Bedürfnisse zu befrieden soll ein Services-Server geschaffen werden, der sich zu einem bestehenden IRC Chat verbindet und diese Dienste anbietet. Die Umsetzung soll in der Programmiersprache C erfolgen. Namentlich soll folgende Dienste angeboten werden:

- **Nickserv:** Dienst für die Registrierung eines Nickname um diesen mittels Passwort vor der Verwendung durch andere zu schützen.
- **Chanserv:** Dienst für die Registrierung eines Channels. Zudem wird die Möglichkeit geschaffen, Operatoren zu registrieren, die sich in einem registrierten Channel von Chanserv die Operatoren Berechtigung selber geben können.
- **Botserv:** Dienst für die Erstellung von Bots die einen Channel offenhalten sollen.
- **Opserv:** Dienst für Benutzer mit erweiterten Rechten (IRC Operatoren, Administratoren).
- **Adminserv:** Dienst für Administratoren um bestimmte Operationen wie z.B. das Speichern der Datenbank vor einem Neustart durchzuführen.

Um keine Abhängigkeit von einem Datenbankserver zu erzeugen sollen die Daten in einer SQLite-Datenbank gespeichert werden. SQLite eine Datenbank die Dateibasiert ist und SQL Syntax unterstützt. Der gesamte Datenbestand soll beim Starten des Servers in den Arbeitsspeicher geladen und in regelmäßigen Abständen gespeichert werden.

Die spezifischen Einstellungen des Servers und der Dienste sollen konfigurierbar sein. Des Weiteren soll für jeden Befehl der gesendet werden kann eine Hilfedatei erstellt werden die den jeweiligen Befehl erklärt.

1.3 Aufgabenstellung

Um die in Kapitel 1.2 genannten Ziele zu erreichen werden folgende Teilschritte durchgeführt:

- Marktanalyse: Feststellung bereits existierender Lösungen
- Studium des IRC Protokolls (RFC 1459) um die protokollarischen Anforderungen zu erfassen.
- Aufnehmen der Anforderungen: Welche Befehle soll ein Dienst unterstützen und wie sollen diese funktionieren?
- Festlegung der Grenzen: wie viele Einträge sollen Listen maximal haben dürfen?
- Erarbeiten der Grundlagen für Timer und Signale welche für Timeouts benötigt werden.
- Durchführung des Requirement Engineering
- Ausfertigung des technischen Entwurfs
- Implementierung Basis-Server
- Implementierung der Dienste
- Testing inkl. Ausarbeitung eines Testkonzepts und Testprotokolls
- Dokumentation

2 Einführung in IRC und Services

2.1 Geschichte des IRC

Das IRC (Internet Relay Chat) stammt noch aus den Anfängen des heutigen Internets. Um 1988 wurde das ARPAnet außer Betrieb genommen und das NSFnet übernahm die Funktionen. Hierdurch wurde das TCP/IP Protokoll erschaffen und auseinander liegende Institutionen konnten miteinander kommunizieren.¹

Die ursprüngliche Idee eines Chat-Netzwerkes entstand im BITNET unter dem Namen Relay Chat. Dieses System wurde vom finnischen Studenten Jarkko Oikarinen, der an der Fakultät für Informatik der Universität Oulu studierte, im Sommer 1988 auf das Internet übertragen.²

Durch einen Freund Oikarinens, erfuhren Universitäten in den Vereinigten Staaten von dem Chat. Es folgte eine Anfrage, ihren Server mit dem finnischen zu verbinden. So verliess der IRC zum ersten Mal Finnland.

Bereits 1989 gingen die ersten deutschen Universitäten mit Chats ans Netz und einige geschichtliche Ereignisse trugen in der Vergangenheit in besonderem Maße zur Popularität des IRC bei. Während des Golfkrieges im Jahr 1991 waren die aktuellsten Berichte über den Verlauf des Krieges über den IRC erhältlich. Ein ähnliches Szenarium spielte sich im September 1993 ab, als gegen das sowjetische Staatsoberhaupt, Boris Yeltsin, geputscht wurde. IRC-Benutzer aus Moskau erzählten live von den dortigen Ereignissen.³

Im Jahr 2000 ging in der Schweiz der Bluewin Chat online. Zu ihren besten Zeiten waren über 6'000 Benutzer gleichzeitig online und bildeten so den populärsten Chat der damaligen Zeit.

Seit Mitte der 2000er Jahre die Flash basierten Chats an Popularität gewannen gingen die Nutzerzahlen auf dem IRC Netzwerken zurück. Die grossen Netzwerke wie zum Beispiel QuakeNet oder FreeNode erfreuen sich jedoch immer noch grosser Beliebtheit.

2.2 Funktionsprinzip des IRC

Mehrere IRC Server können sich zu einem Netzwerk zusammenschliessen. So können die Benutzer die Benutzer von allen Servern miteinander agieren.

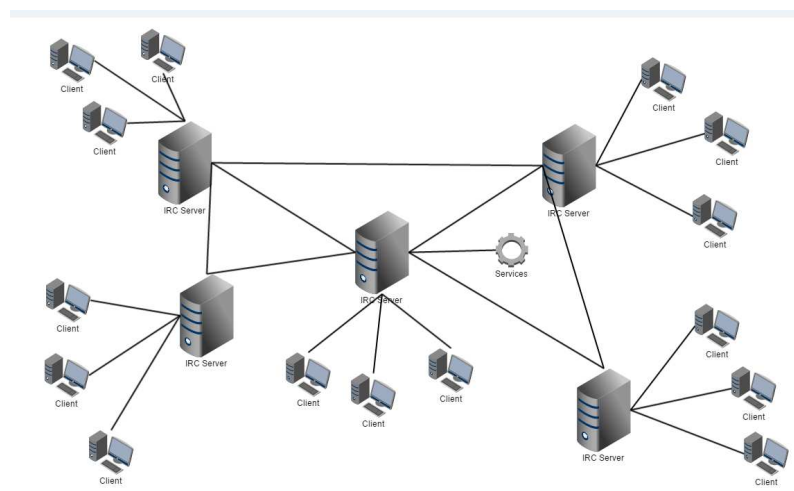


Abb. 2: Schema eines IRC Netzwerkes

Wie in Abb. 2 zu sehen ist, verbinden sich die Clients jeweils zu einem Server. Die Server schliessen sich zu einem Netzwerk zusammen. Jeder Benutzer kann so alle anderen Benutzer erreichen. Das Gleiche gilt für die Services. Die Services sind im Prinzip nichts anderes als ein weiterer Server, der sich mit einem der im sich Netzwerk befindlichen Servern verbindet.

Damit ein Benutzer chatten kann benötigt er einen Client. Diese sind in der Regel kostenlos erhältlich. Populäre Clients sind z.B. mIRC für Windows oder XChat für Linux. Das eigentliche Chatten geschieht über das Senden von Befehlen. Möchte ein Benutzer eine Nachricht an einen anderen Benutzer schicken, so tippt er z.B. /msg Benutzer Nachricht

Der Slash (/) signalisiert dem Client so, dass ein Befehl abgesetzt werden soll. Er wandelt den erhaltenen Befehl dann in das Format um, das vom Protokoll verlangt wird. Die genaue Syntax für die Eingabe kann von Client zu Client variieren, die endgültige Form wird jedoch vom Protokoll definiert.

2.3 Protokoll

Um eine einheitliche Form der Nachrichtenübermittlung zu gewährleisten sind die IRC Befehle in einem Protokoll genau definiert.

Somit wird sichergestellt, dass die Nachrichtenübermittlung mit jedem Client und jedem Server in einer strukturierten Art und Weise zuverlässig funktioniert.

Das Protokoll für IRC ist als RFC 1459 (Request for Comments) definiert. Mit einem RFC gibt man ein Dokument für eine Diskussion frei. Es behält den Namen RFC auch dann, wenn es allgemeine Akzeptanz erreicht hat.

Nachfolgend wollen wir einige wichtige Punkte des Protokolls betrachten.

2.3.1 Einführung

2.3.1.1 Grundlagen

Das Protokoll legt fest, dass jegliche Nachrichten die über das Netzwerk gesendet werden, aus den drei Teilen

- Präfix (optional)
- Befehl
- Parameter

bestehen und folgendes Format aufweisen müssen:

```
:Präfix Befehl Argument CRLF
```

Zwischen dem Kolon (:) und dem Präfix darf kein Leerzeichen stehen und jeder Befehl muss mit Carriage Return (CR) und Line Feed (LF) abgeschlossen werden.

Der Befehl muss ein gültiger IRC Befehl oder dessen numerische Repräsentation sein. Eine Liste gültiger Befehle kann auf <https://tools.ietf.org/html/rfc1459>.

Wir dieses Format nicht eingehalten, wird der Befehl vom Server ignoriert. Auf die meisten Befehle soll der Server eine Antwort senden. Diese hat eine eindeutige Nummer und ist im Kapitel 6 des Protokolls definiert.

2.3.1.2 Operatoren

Um eine grundlegende Ordnung aufrecht zu erhalten kann der Betreiber des IRC Servers sogenannte Operatoren ernennen. Die haben im Vergleich zu anderen Benutzern erweiterte Rechte. Diese sind nicht mit Operatoren in Chaträumen zu verwechseln, welche Störenfriede aus dem Chatraum verbannen können. Vielmehr haben diese Operatoren Rechte die für den ganzen Server gelten. So können diese zum Beispiel einen Benutzer von gesamten Netzwerk verbannen.

2.3.1.3 Channels

Ein Channel ist ein Chatraum. Also ein Gebilde in dem sich mehrere Benutzer treffen und austauschen können. Nachrichten die an einen Channel gesendet werden können von allen Benutzern in diesem Channel gelesen werden. Einen Channel kann man mit dem Befehl JOIN betreten.

Wie in Kapitel 2.3.1.2 erwähnt gibt es neben den Server-Operatoren auch Channel-Operatoren die unerwünschte Benutzer aus dem Channel verbannen können. Diese Operatoren haben auch weitere Rechte, zum Beispiel können diese auch weitere Benutzer zu Operatoren ernennen.

2.3.2 Die IRC Spezifikation

Für IRC wurde kein spezifischer Zeichensatz definiert. Vorgegeben ist lediglich, dass ein 8-Bit Zeichensatz verwendet muss. Jedoch hat sich das amerikanische ASCII als Standard durchgesetzt. Das heisst, dass in der Regel keine Umlaute für Server Befehle verwendet können. Dasselbe gilt für Benutzernamen, sogenannte Nicknames.

Die Kommunikation zwischen den Benutzer wird hingegen nicht beeinträchtigt. Dort können jedwede Art von Zeichen gesendet werden, da die meisten Server für die Kommunikation UTF-8 unterstützen.

2.3.3 IRC Konzepte

In diesem Kapitel untersuchen wir die protokollarischen Konzepte der Nachrichtenübermittlung basierend auf dem nachfolgenden Bild.

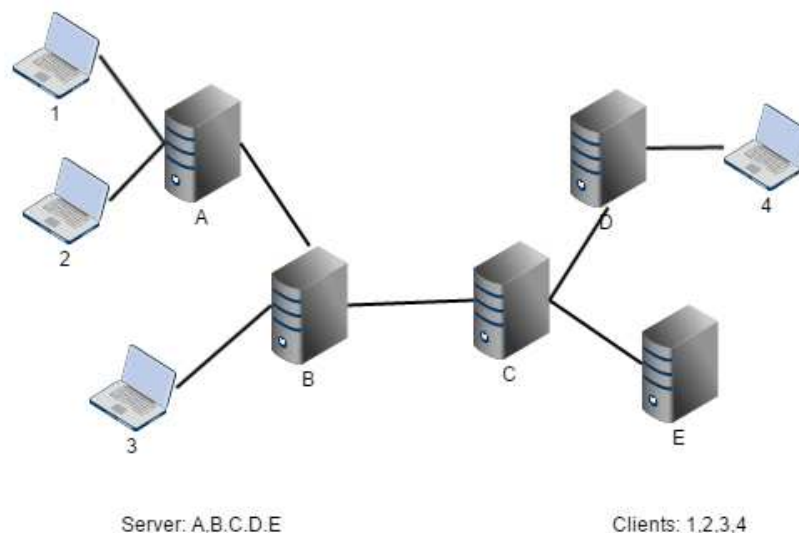


Abb. 3: Struktur eines IRC Netzwerks

2.3.3.1 1:1 Kommunikation

Die Kommunikation auf einer 1:1 Basis erfolgt in der Regel zwischen Benutzern, da die Server nicht ausschliesslich untereinander kommunizieren.

Der Pfad einer Nachricht nimmt stets den kürzesten zwischen zwei Punkten im Netzwerk. 1:1 Nachrichten werden nur von Sender, Empfänger und den Servern, die die Nachricht weiterleiten sollen gesehen.

Beispiel 1: Eine Nachricht zwischen Client 1 und 2 wird nur vom Server A gesehen, der diese direkt an Client 2 schickt.

Beispiel 2: Eine Nachricht zwischen Client 1 und 3 wird von den Server A und B gesehen.

Beispiel 3: Eine Nachricht zwischen Client 2 und 4 wird von den Servern A, B, C und D gesehen.

2.3.3.2 1:n Kommunikation

Die 1:n Kommunikation auf dem IRC tritt in verschiedenen Formen auf. Diese sind:

- Nachricht an eine Liste: Der Client schickt die Nachricht an eine Liste von Empfängern.
- Nachricht an eine Gruppe: Der Client schickt eine Nachricht einen Channel
- Nachricht an einen Host: Der Client schickt eine Nachricht Empfänger welche über die gleiche Hostmaske verfügen. Beispiel: Eine Nachricht an `*@*.swisscom.com` empfangen alle, die über einen Swisscom-Anschluss auf dem Server verbunden sind. Diese Art der Nachricht ist üblicherweise Operatoren vorbehalten.

2.3.3.3 1:Alle Kommunikation

Diese Art der Kommunikation beinhaltet Nachrichten die an alle Clients oder Server oder beides verschickt werden. Dies kann ein hohes Datenvolumen verursachen.

2.3.3 Nachrichten Details

Damit sich ein Client zu einer Server verbinden kann, muss sich der Client nach Erstellung der Verbindung mit dem Server registrieren. Dies geschieht über die Übermittlung von Verbindungsnachrichten. Die nachfolgenden Nachrichten müssen demnach gesendet werden.

2.3.3.1 Pass Nachricht

Diese Nachricht ist nur dann erforderlich, wenn der Server mit einem Passwort geschützt ist. Folgende Syntax ist zwingend:

```
PASS <Passwort> CRLF
```

2.3.3.2 Nick Nachricht

Mit diesem Befehl legt man seinen eigenen Nicknamen fest.

```
NICK <Nickname> CRLF
```

Wird der Nickname bereits verwendet meldet der Server die Fehlermeldung

```
ERR_NICKNAMEINUSE
```

zurück.

2.3.3.3 User Nachricht

Mit dieser Nachricht werden die Details des Client an den Server gemeldet.

```
USER <Benutzername> <Hostname> <Servername> <Realname>
```

Der Benutzername ist Teil der Hostmaske. Diese wird in den Einstellungen im Programm zum Chatten in der Regel als E-Mail Adresse angegeben und dem Server so übermittelt.

Der Hostname wird durch den Internetanbieter bestimmt. Er ist von der IP Adresse abstrahiert.

Der Servername wird automatisch gesetzt. Verbindet man sich auf ein Netzwerk mit mehreren Servern ist hier der Name des Servers über den man sich verbindet gemeint.

Der Realname kann frei festgelegt werden. Er kann öffentlich eingesehen werden, daher sollte man sich gut überlegen, ob man seinen echten Namen angeben will.

Als Beispiel können wir den Windows Client mIRC hinzuziehen. Bevor man sich auf einen Chat verbindet muss man in den Einstellungen diese Informationen hinterlegen:

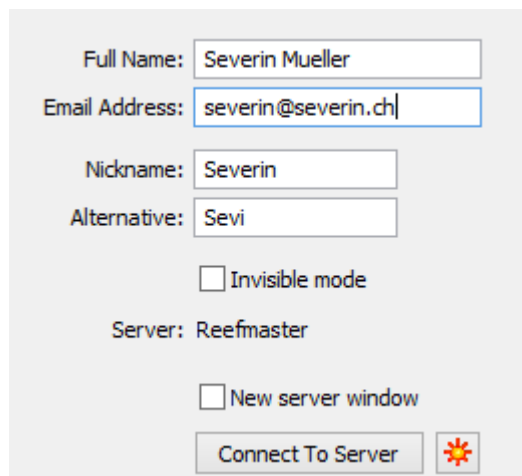


Abb. 4: Verbindungseinstellungen mIRC

Der Full Name ist der Realname und die E-Mail Adresse liefert den Benutzername. Jedoch hat nur der Teil vor dem @ einen Einfluss.

Wie erwähnt können wir den Hostnamen nicht beeinflussen. Wenn uns diese Information interessieren sollte können wir nach dem Verbinden eine WHOIS Nachricht an den Server schicken. Mit WHOIS werden diverse Information über den angegebenen Benutzer angezeigt. Syntax

```
WHOIS <Nickname>
```

Wenn wir also im Beispiel Severin den Hostnamen herausfinden möchten, senden wir die Nachricht /whois Severin

an den Server. Folgendes kommt zurück:

```
[16:57:31] * Severin is connecting from *@185-103.60-188.cust.bluewin.ch 188.60.103.185
[16:57:31] * /whois Info Severin
[16:57:31] * Severin is ~severin@185-103.60-188.cust.bluewin.ch
[16:57:31] * Realname: Severin Mueller
[16:57:31] * Severin on #c-unregistered
[16:57:31] * Severin using sendak.freenode.net, Vilnius, Lithuania, EU
[16:57:31] * Severin has been idle 11secs, signed on 6secs ago
[16:57:31] * Severin End of /WHOIS Info.
[16:57:31] -
```

Abb. 5: Rückgabe einer WHOIS Nachricht

Der Hostname ist hier gelb markiert.

2.3.3.4 Server Nachricht

Wollen wir statt einen Client einen Server verbinden, so müssen wir eine Server Nachricht schicken.

```
SERVER <Servername> <Hopcount> <Info> CRLF
```

Servername ist der Name des Servers den wir mit dem Netzwerk verbinden wollen.

Hopcount ist eine interne Information die indiziert, wie weit die Server voneinander weg sind (wie viele Knoten dazwischen liegen)

Info ist eine Beschreibung des Servers.

2.3.3.5 PRIVMSG

Um eine Nachricht an einen Client oder Server zu senden, verwenden wir PRIVMSG.

```
PRIVMSG Empfänger :Nachricht CRLF
```

Auch Dienste, die in diesem Projekt implementiert werden, werden mit dieser Nachricht angesteuert.

2.3.3.6 NOTICE

Eine weitere Möglichkeit eine private Nachricht zu senden ist NOTICE. Üblicherweise wird in diesem Fall kein neues Chat-Fenster geöffnet, sondern die Nachricht wird im aktiven Fenster angezeigt.

```
NOTICE Empfänger :Nachricht CRLF
```

Dienste wie in diesem Projekt senden ihre Antworten üblicherweise als NOTICE.

Auf weitere Details des Protokolls verzichten wir an dieser Stelle. In den späteren Kapiteln dieser Arbeit werden wir auf die benötigten Teile des Protokolls eingehen.

2.4 Services

Wie in Kapitel 1 erwähnt bietet das IRC Protokoll an sich keinerlei Funktionen um einen Nicknamen oder Channel zu reservieren. Für diesen Zweck wurden Services geschaffen.

Nachfolgend möchten wir erläutern, was genau wir unter diesen Services verstehen.

2.4.1 Einführung

Services sind im Grunde nichts anderes als ein Server der sich zum IRC Netzwerk verbindet. Auf diesem Server laufen die einzelnen Dienste wie normale Clients, mit dem Unterschied, dass diese eine Kennung besitzen, die sie als Service kennzeichnet. Diese Kennung wird als sogenannter Usermode vom Server zur Verfügung gestellt (siehe Kapitel 4.2.3.2 im IRC Protokoll).

Es gibt keine allgemeingültige Definition welche Services ein Server unterstützen soll. Das Protokoll legt lediglich fest, dass ein Benutzer als Service markiert werden kann.

In der Praxis haben sich jedoch einige Arten von Services durchgesetzt. Dazu zählen:

- Nickserv: Dienst um Nicknames zu registrieren
- Chanserv: Dienst um Channels zu registrieren
- Memoserv: Dienst um Nachrichten an registrierte Benutzer zu senden
- Opserv: Dienst für Server-Operatoren
- Botserv: Dienst für das Erstellen von Bot die als eine Roboter fungieren
- Hostserv: Dienst um für bestimmte Benutzer neue Hosts zu setzen
- Adminserv: Dienst für Server Administratoren

In dieser Arbeit befassen wir uns nur mit Nickserv, Chanserv, Opserv, Botserv und Adminserv.

2.4.2 Nickserv

Wie erwähnt stellt Nickserv einen Dienst zur Verfügung mit dem man sich einen gewünschten Benutzernamen registrieren kann. Üblicherweise wird dieser mit einem persönlichen Passwort geschützt. Da heisst, dass man sich für diesen Nick identifizieren muss, bevor man ihn verwenden kann. In der Regel werden weitere Funktionen zur Verfügung gestellt. Meist sind dies Optionen die mit anderen Diensten interagieren.

2.4.3 Chanserv

Chanserv oder Channel Services sind ein Dienst für die Registrierung von Channels. Mit einer Registrierung soll sichergestellt werden, dass unbefugte Benutzer den Channel nicht „kapern“ können. Unter kapern verstehen wir das erschleichen von Op-Rechten um dann den Channel zu übernehmen. Da das IRC Protokoll vorsieht, dass jeder Benutzer einem anderen Benutzer Op-Rechte geben kann kam dies häufig vor. Mittels einer Registrierung kann sich der Eigentümer des Channels jederzeit über Chanserv selber die Op-Rechte zurückgeben lassen und so sicherstellen, dass der Channel nicht gekapert wird.

2.4.4 Opserv

Opserv soll Server Operatoren die Möglichkeit geben, einige Operation global, also auf allen Server des Netzwerkes zu steuern. Wir wollen einen Schritt weitergehen und Administratoren die Möglichkeit geben, normalen Benutzern ohne Operatoren Recht Zugriff auf Opserv zu erlauben. Dies kann z.B. nützlich sein, wenn kein Operator anwesend ist und man einen Benutzer dem man vertraut die Rechte geben kann damit dieser vorläufig nach dem Rechten sieht.

2.4.5 Botserv

Es kann vorkommen, dass auf gewissen Netzwerk Channels die nicht registriert wurden eine Möglichkeit geben möchte, ein Takeover (kapern) des Channels zu verhindern, ohne dass der Channel registriert werden muss. Mit Botserv kann man sogenannte Bots erstellen, die die Aufgabe haben, einen Channel offenzuhalten und ein Takeover zu verhindern. Ein Bot wird mit einem Passwort versehen, mit welchen man sich für den Bot identifizieren kann und so auf die Funktionen Zugriff erhält.

2.4.6. Adminserv

Mit Adminserv kann man Administratoren erlauben, gewisse Server Operationen durchzuführen. Diese können z.B. ein Neustart der Services, das Speichern der Datenbanken oder das Abschalten der Dienste sein.

Die genaue Implementierung kann sich zwischen den einzelnen Lösungen stark unterscheiden. Wir möchten daher untersuchen, welche Lösungen bereits existieren wo es Lücken gibt, die es zu schließen gilt.

2.4.7 Festlegung der Grenzen

Um eine gewisse Stabilität und Performance zu gewährleisten müssen wir einige Grenzen festlegen. Folgende Limitationen sind zu berücksichtigen:

- Die Befehlslänge wird auf 1024 Zeichen begrenzt
- Die Länge von Nickname und Channels wird durch das Protokoll auf 32 Zeichen begrenzt
- Das Projekt ist zunächst nur für Linux zu implementieren

3 Marktanalyse

Wie in der Computerwelt üblich gibt es auch für IRC Services und Server verschiedene Lösungen. Einige davon sind proprietäre, andere Open-Source Lösungen. Wir möchten in diesem Kapitel einige davon genauer betrachten und feststellen, wo unsere Lösung bestimmte Lücken füllen kann.

3.1 Webmaster ConferenceRoom

ConferenceRoom von Webmaster ist eine proprietäre Lösung die eine All-in-One Lösung bietet. Das heisst, dass der IRC Server und die Services in einem Produkt enthalten sind.

Der Vorteil dieser Lösung liegt in den integrierten Services und der Stabilität der Software.

Der Nachteil sind die eher hohen Kosten. Eine Version für 1000 Benutzer kostet \$250.00 und die Version für 10000 Benutzer kostet \$995.00⁴

Die 1000 Benutzer Version bietet zudem nur die Dienste Nickserv und Chanserv an. Möchte man Memoserv dazu haben benötigt man die teurere Version.

Der Chat von Bluewin wurde auf einem Conference Room Server betrieben.

3.2 Unreal IRCd

Unreal IRCd ist ein Open Source IRC Server der zu den populärsten Servern überhaupt gehört. Er läuft äusserst stabil, bietet eine Vielzahl an Konfigurationsmöglichkeiten und ist aussergewöhnlich gut dokumentiert. Als Open-Source Lösung ist der Server kostenlos erhältlich. Wenn man den Server jedoch modifiziert darf man keine technische Unterstützung erwartet.

Ein Nachteil dieser Lösung ist, dass keine Services mitgeliefert werden. Diesem Problem kann man begegnen indem man auf eine externe Lösung zurückgreift. Nachfolgend betrachten wir einige Open-Source Services.

3.3 IRC Services

Die IRC Services sind quasi das Original. Praktisch alle populären Lösungen basieren auf dem original Source Code von Andrew Church. Die erste Version wurde 1996 entwickelt. Diese Service werden nicht mehr weiterentwickelt. Die allererste Version dieser Lösung ist immer noch online auf <http://achurch.org/services/> zu finden.

Da die Entwicklung vor einiger Zeit eingestellt wurde bieten diese Services nicht den Umfang die moderne Systeme bieten.

3.4. Anope Services

Anope ist eine Services Lösung die auf den originalen IRC Services basiert. Anope bietet eine Vielzahl an Optionen und läuft äusserst stabil. Anope verfügt über eine gute Dokumentation und 6 Services. Zudem werden Module, die einfach dazu geschaltet werden können, unterstützt.

Der Funktionsumfang von Anope ist beträchtlich.

Ein Nachteil ist jedoch, dass sich die Syntax und die Ausgaben teilweise stark von denen die ConferenceRoom bieten unterscheiden und ehemalige Swisscom-Chatter Mühe haben, sich zurecht zu finden.

3.5 Auspice

Auspice sind eine Open-Source Lösung die auch unter Windows funktioniert. Leider wurde die Entwicklung 2005 eingestellt. Sie sind aber nach wie vor zum Download erhältlich und auch in Betrieb.

3.6 Fazit

An den heute erhältlichen IRC Services ist zu sehen, dass IRC an Popularität eingebüsst hat. ConferenceRoom und Anope werden weiterentwickelt, genau wie auch Unreal IRCd, die weiteren Services werden nicht mehr wirklich gepflegt.

Da viele der ehemaligen Swisscom Chatter sich wünschen, dass die Services ähnlich aufgebaut sind wie diejenigen von ConferenceRoom soll unsere Lösung dieses Ziel verfolgen.

4 Anforderungen

In diesem Kapitel wollen wir die Anforderungen für die Services erfassen. Zunächst wollen wir die Use Cases beschreiben. Aufgrund der Use Cases erstellen wir die Anforderungen. Als Akteure sollen die Stakeholder dienen die wir wie folgt definieren.

4.1 Basis-Server

Wir definieren die Akteure für den Basis-Server wie folgt

Akteur	Beschreibung
Server Administrator	Kann den Server konfigurieren und verfügt über alle Rechte
Basis-Server	Der Basis auf dem die Services laufen. Verbindet sich zum IRC Server

Tabelle 1: Stakeholder

Als Basis Server wird der eigentlich Server bezeichnet, auf dem die Services laufen sollen. Der Server soll konfigurierbar sein, demnach muss das Programm einen Mechanismus zur Konfiguration bereitstellen.

4.1.1 Use Case Basis Server (UC S-01)

Use Case Basis-Server

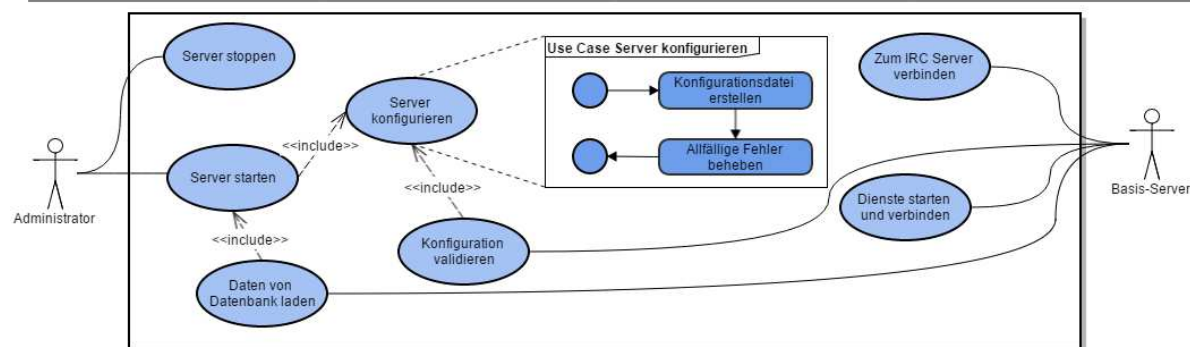


Abb. 6: Use Case Basis Server

Die Benutzer schicken ihre Nachrichten an den IRC Server und nicht an den Services-Server. Der IRC Server leitet dann die Nachricht an den richtigen Service weiter.

4.1.2 Anforderungen Basis-Server

4.1.2.1 IRC Server

Zunächst müssen wir festlegen, welche Chat-Server überhaupt unterstützt werden sollen. Die Entscheidung fiel auf den Unreal IRCd Server, da diese Lösung die populärste ist und über eine gute dokumentierte Unterstützung für Services-Schnittstellen verfügt. Zu beachten ist dabei, dass das Unreal IRCd wie alle Server eine eigene Protokoll-Implementierung nutzen. Für sämtliche Nachrichten an den Server ist dies zu beachten. Das IRCd-Protokoll im Projektordner „unreal“ zu finden.

Daraus resultiert folgende Anforderung:

Requirement Nr.	R-S-001
Titel	Unterstützte IRC Server
Beschreibung	Der Unreal IRCd Server soll unsere Services unterstützen. Es sind die Grundlagen zu schaffen, dass der Basis-Server sich zu einem bestehenden Unreal IRCd Server verbinden kann.
Begründung	Unreal IRCd ist der populärste Open-Source IRC Server. So werden die Services einem breiten Publikum angeboten
Kommentare	

Tabelle 2: R-S-001: Unterstützte IRC Server

4.1.2.2 Konfiguration Basis-Server

Die Services sollen einen hohen Grad an Flexibilität aufweisen. Daher ist es erforderlich, dass alle wichtigen Werte des Basis-Servers konfiguriert werden können.

Requirement Nr.	R-S-002
Titel	Konfiguration Basis-Server
Beschreibung	Die Grundeinstellungen des Basis-Servers sollen konfigurierbar sein. Namentlich sind dies: Name des Basis-Servers Beschreibung Username und Hostname Post Adresse des IRC Servers Name des IRC Servers Passwort für die Verbindung Die Konfiguration soll als Text-Datei vorliegen.
Kommentare	

Tabelle 3: R-S-002: Konfiguration Basis-Server

4.1.2.3 Starten und Stoppen des Servers

Das Starten und Stoppen des Servers soll über ein Argument auf der Kommandozeile erfolgen.

Requirement Nr.	R-S-003
Titel	Starten und Stoppen des Servers
Beschreibung	Die Services sollen von der Kommandozeile gestartet und gestoppt werden können. Als Argument für den Programmstart wird „start“ festgelegt Als Argument für das Stoppen wird „stop“ festgelegt.
Kommentare	

Tabelle 4: R-S-003: Starten des Servers

4.1.2.4 Validierung Server-Konfiguration

Da es wichtig ist, dass in der Konfiguration nur gültige Werte stehen soll das Programm beim Start einen Fehler ausgeben wenn die Konfiguration nicht ordnungsgemäss validiert werden kann.

Requirement Nr.	R-S-004
Titel	Validierung Server-Konfiguration
Beschreibung	Der Server soll die Konfiguration beim Programmstart validieren und einen Fehler im Falle ungültiger Werte ausgeben. Die Zeilennummer der Konfiguration und eine aussagekräftige Fehlermeldung sind auszugeben.

Kommentare	
------------	--

Tabelle 5: R-S-004: Validierung Server-Konfiguration

4.1.2.5 Daten von Datenbank laden

Aus Performancegründen sollen die Daten im Arbeitsspeicher gehalten werden. Daher müssen die Daten beim Programmstart von der Datenbank in den Arbeitsspeicher geladen werden.

Requirement Nr.	R-S-005
Titel	Validierung Server-Konfiguration
Beschreibung	Der Server soll sämtliche Daten beim Programmstart von der Datenbank mittels SQL Query aus der SQLite Datei lesen und in den Arbeitsspeicher laden.
Kommentare	

Tabelle 6: R-S-005: Daten von Datenbank laden

4.1.2.6 Verbindung zum IRC Server

Der Basis-Server soll automatisch die Verbindung zum IRC Server erstellen. Dies soll aufgrund der Konfiguration erfolgen, ohne dass der Administrator weitere Eingaben tätigen muss.

Requirement Nr.	R-S-006
Titel	Verbindung zum IRC Server erstellen
Beschreibung	Der Server soll automatisch die Verbindung zum IRC Server erstellen. Dafür muss zunächst ein Socket geöffnet und eine Verbindung erstellt werden. Danach müssen die vom Protokoll verlangten Nachrichten im korrekten Format aufbereitet und an den Socket gesendet werden. Sind alle Nachrichten korrekt übertragen worden soll die Verbindung bis zum Programmende stehen. Tritt ein Fehler auf, soll dieser entsprechend ausgegeben und das Programm beendet werden.
Kommentare	

Tabelle 7: R-S-006: Verbindung zum IRC Server

4.1.2.7 Dienste starten und verbinden

Requirement Nr.	R-S-007
Titel	Dienste starten und verbinden
Beschreibung	Da die Dienste im Prinzip spezielle Clients sind, sind diese wie normale Benutzer auf dem Server zu registrieren. Die entsprechende Nachricht muss aufbereitet und an den Server gesendet werden. Siehe dazu Kapitel 3.1 im IRC Protokoll
Kommentare	

Tabelle 8: R-S-007: Dienste starten und Verbinden

4.1.2.8 Daten in regelmässigen Abständen speichern

Sobald die Verbindung zum IRC Server steht müssen die einzelnen Dienste gestartet und verbunden werden.

Requirement Nr.	R-S-008
Titel	Daten in regelmässigen Abständen speichern
Beschreibung	Die Daten sollen im in der Konfiguration festgelegten Intervall in der Datenbank gespeichert werden.

	Dazu ist ein Mechanismus zu erschaffen, der alle zwei Sekunden ein Signal an den Server schickt, damit dieser jedwede Art von Intervallen überprüfen kann.
Kommentare	

Tabelle 9: R-S-008: Daten in regelmässigen Abständen speichern

4.1.2 Dienste allgemein

Jeder Dienst muss eine Reihe von Befehlen unterstützen. Um einen solchen Befehl auszuführen zu können muss der Benutzer in seinem Chatclient den Befehl als private Nachricht an den jeweiligen Dienst senden (siehe auch Kap. 2.3.3.5).

Dienste müssen ihre Antworten als NOTICE zurücksenden (siehe Kapitel 2.3.3.6).

Für sämtliche Anforderung die nachfolgend erfasst werden gilt für die Befehlssyntax:

<>: Optionale Parameter, Text

[]: Vordefinierter Parameter (Befehl)

4.1.3 Datenbank

Um die anfallenden Daten zu speichern benötigen wir eine Datenbank. Um Abhängigkeiten zu vermeiden verwenden wir eine SQLITE Datenbank, da keine separate Software installiert werden muss um diese zusammen mit dieser Applikation zu verwenden. Die Datenbank soll alle 20 Minuten automatisch gespeichert werden.

4.1.4 Logging

Sämtlicher Datenverkehr der über den Basis-Server läuft soll geloggt werden. Zusätzlich sollen die Start und Ende sämtlicher Datenbankfunktionen geloggt werden um das debuggen zu erleichtert. Ebenso sollen sämtliche Sqlite Fehlermeldungen in das Log geschrieben werden. Es ist daher eine Funktion zu bauen, mit der man einen Logeintrag der Kategorie DEBUG, WARN oder ERROR geschrieben werden kann.

4.2 Nickserv

Wir definieren die Stakeholder für Nickserv wie folgt:

Stakeholder	Beschreibung
Server Administrator	Kann Konfigurationen für Nickserv vornehmen und hat Vollzugriff auf alle Befehle. Das bedeutet, dass er Passwörter für beliebige Nicknames einsehen und neu setzen kann.
Benutzer	Kann Nickserv nutzen. Einige Befehle erfordern keine Berechtigung, die meisten jedoch schon.

Tabelle 10: Stakeholder Nickserv

Als weiteren Akteur definieren wir:

Akteur	Beschreibung
Nickserv	Empfängt Befehle von Nutzer und Administratoren, verarbeitet diese und liefert immer eine Antwort zurück.

Tabelle 11: Akteure Nickserv

4.2.1 Use Case Nickserv (UC NS-01)

Use Case Nickserv

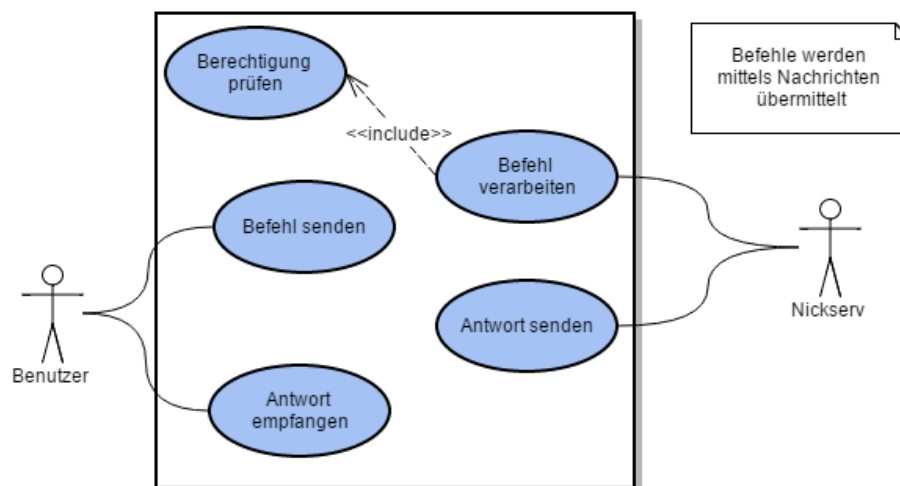


Abb. 7: Use Case Nickserv

4.2.2 Anforderungen Nickserv

Folgende Funktionen soll Nickserv anbieten

- **ACC** – Feststellung mit welcher Berechtigung ein Benutzer einen Nickname verwendet.
- **ACCESS** – Eine Liste in welche Benutzeradressen eingetragen werden können, sodass der Benutzer sich nicht explizit identifizieren muss, um den Nickname zu verwenden.
- **AUTH** – Ein Mechanismus mit der ein Benutzer explizit zustimmen muss, um zu Operator oder Notifylisten hinzugefügt zu werden.
- **DROP** – Registrierung eines Nickname aufheben
- **GETPASS** – Passwort eines Nicknames einsehen
- **GHOST** – Im Falle eines Ping-Timeout, also den Verlust der Verbindung kann es vorkommen, dass der Nickname trotzdem noch online ist (sogenannter Ghost). Diese Verbindung soll getrennt werden können.

- **IDENTIFY** – Identifikation für einen Nickname
- **INFO** – Zeigt Information über einen Nickname an
- **LIST** – Zeigt eine Liste von registrierten Nicknames an.
- **LISTCHANS** – Zeigt eine Liste von Channels in denen ein Nickname spezielle Rechte hat.
- **NOTIFY** – Eine Liste von „Freunden“. Verbindet sich ein Freund zum Server, sendet Nickserv eine Information.
- **REGISTER** – Einen Nickname registrieren.
- **RELEASE** – Wird ein Nickname ohne Berechtigung verwendet, so sperrt Nickserv die weitere Verwendung. Mit RELEASE kann diese Sperre aufgehoben werden.
- **SET** – Einstellungen für den Nickname. Diese sind:
 - **AUTHORIZE** – Stellt ein, ob und für welche Liste die explizite Zustimmung gegeben werden muss.
 - **EMAIL** – Legt die E-Mail Adresse fest
 - **HIDEEMAIL** – Legt fest, ob die E-Mail Adresse in der Nick Information angezeigt wird.
 - **MFORWARD** – Legt eine automatische Weiterleitung von Memos fest (Memoserv noch nicht implementiert)
 - **MLOCK** – Legt Benutzermodi fest, die automatisch nach der Identifikation gesetzt werden.
 - **NOMEMO** – Legt fest, ob ein Benutzer Memos erhalten will (Memoserv noch nicht implementiert)
 - **NOOP** – Legt fest, ob der Benutzer in Channel in denen er Operatoren-Rechte hat, automatisch diesen Status erhalten soll.
 - **PASSWORD** – Legt ein neues Passwort für den Nickname fest.
 - **PROTECT** – Legt den Schutz für den Nickname fest. Es stehen drei Schutz-Stufen zur Verfügung
 - **URL** – Legt eine URL fest, die in der Nick Info angezeigt wird.
- **SETPASS** – Legt ein neues Passwort für einen Nickname fest.

Die Funktionen müssen die nachfolgend definierten Fähigkeiten aufweisen.

4.2.3 Anforderung Nickserv Hauptfunktion

Requirement Nr.	R-NS-001
Titel	Nickserv Hauptfunktion
Beschreibung	Es ist Handler für private Nachrichten an Nickserv zu implementieren, welche als PRIVMSG an Nickserv gesendet werden. Die Nachricht muss im Format source PRIVMSG Nickserv :Nachricht vorliegen. Nickserv muss dann anhand einer dynamischen Befehlsliste automatisch aufgrund der Nachricht den korrekten Befehl aufrufen. Beinhaltet Nachricht keinen gültigen Befehl wird eine entsprechende Fehlermeldung zurückgegeben.
Kommentare	

Tabelle 12: R-NS-001: Nickserv Hauptfunktion

4.2.4 Konfiguration Nickserv

Requirement Nr.	R-NS-002
Titel	Konfiguration Nickserv
Beschreibung	Folgende Konfigurationsmöglichkeiten soll Nickserv anbieten: Abschnitt „general“

	<p>enabled: Nickserv aktivieren name: Nickname von Nickserv realname: Realname von Nickserv expiry: Zeit in Tagen wann der Nickname bei Nichtbenutzung gelöscht wird admin: Welchen Administrator Level ein Benutzer für den Vollzugriff benötigt. Diese sind: 1: Help Operator 2: IRC Operator 3: Co Administrator 4: Server Administrator 5: Services Administrator 6: Network Administrator</p> <p>Abschnitt „registration“ delay: Wie viele Sekunden vergangen sein müssen, bevor erneut ein Nickname registriert werden kann access: Welchen IRC Operator Level ein Benutzer haben muss, um einen Nickname zu registrieren. autoaccess: Legt fest, ob die Adresse des Benutzers automatisch in die ACCESS Liste des Nickname eingetragen werden soll.</p> <p>Abschnitt „list“ maxlist: Max. Anzahl Einträge die beim LIST Befehl angezeigt werden soll operonly: Nur IRC Operatoren können diesen Befehl nutzen</p> <p>Abschnitt „password“</p> <p>getpass: Benötigter Level um diesen Befehl zu nutzen setpass: Benötigter Level um diesen Befehl zu nutzen enforcer: Benutzername bei Sperre eines Nickname release: Zeit in Sekunden bevor ein gesperrter Nickname wieder freigegeben werden soll</p> <p>Abschnitt „default“</p> <p>Legt die Standardeinstellungen fest, die bei der Registrierung eines Nickname gesetzt werden sollen:</p> <p>noop: Nickname soll nicht automatisch Operator Status erhalten. high_protect: Setzt die höchste Schutzstufe hide_email: E-Mail wird in Info nicht angezeigt no_memo: Benutzer erhält keine Memos auth_channel: Benutzer muss zustimmen bevor er auf Operator Listen gesetzt werden kann. auth_notify: Benutzer muss zustimmen, bevor er auf Notify listen gesetzt werden kann. mnotify: Erhält eine Nachricht wenn ein neues Memo eintrifft mlock: Benutzermodi die bei der Identifizierung gesetzt werden sollen.</p>
Kommentare	

Tabelle 13: R-NS-002: Konfiguration Nickserv

4.2.5 Anforderungen Nickserv Unterfunktionen

4.2.5.1 Anforderungen ACC

Requirement Nr.	R-NS-003
Titel	ACC
Beschreibung	<p>Es soll ein Befehl implementiert der dem Absender die Berechtigung, die der angegebene Nutzer für den verwendeten Nickname hat, zurückgibt. Folgendes soll zurückgegeben werden:</p> <p>Hinweis, dass der Nickname nicht registriert ist, falls zutreffend</p> <p>0, falls der Nutzer keinerlei Zugriff auf den Nickname hat</p> <p>1, falls der Nutzer einen passenden Eintrag in der ACCESS List des Nickname hat</p> <p>2, falls der Nutzer sich mittels Passwort für den Nickname identifiziert hat.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ACC <Nickname></p>
Kommentare	<p>Keine besonderen Rechte erforderlich</p> <p>Implementierung von ACCESS ist erforderlich.</p>

Tabelle 14: R-NS-003: ACC

4.2.5.2 Anforderungen ACCESS

Requirement Nr.	R-NS-004
Titel	ACCESS
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Einträge von Benutzeradressen verwaltet werden können. Diese Benutzeradressen sollen den entsprechenden Nickname verwenden können, ohne dass sich der Benutzer explizit für den Nickname identifizieren muss.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu</p> <p>DEL – Entfernt einen Eintrag aus der Liste</p> <p>LIST – Zeigt alle Einträge der Liste an</p> <p>WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ACCESS [Befehl] <Adresse Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Um den Parameter <Nickname> zu nutzen muss der Benutzer Vollzugriff auf Nickserv haben.</p>

Tabelle 15: R-NS-004: ACCESS

4.2.5.3 Anforderungen AUTH

Requirement Nr.	R-NS-005
Titel	AUTH
Beschreibung	<p>Es soll ein Mechanismus geschaffen werden, mit der ein Benutzer seine Zustimmung geben kann, bevor er auf Listen gesetzt werden kann.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ACCEPT: Bestätigen eine Anfrage DECLINE: Ablehnen einer Anfrage READ: Eine Anfrage lesen LIST: Anfragen anzeigen.</p> <p>Die Befehlssyntax wird wie folgt definiert: ACCESS [Befehl] <Adresse Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>AUTHORIZE muss aktiviert sein. .</p>

Tabelle 16: R-NS-005: AUTH

4.2.5.4 Anforderungen DROP

Requirement Nr.	R-NS-006
Titel	DROP
Beschreibung	<p>Ein Nickname soll gelöscht werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert: DROP <Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Benutzer mit Vollzugriff können jeden Nickname löschen.</p>

Tabelle 17: R-NS-006: DROP

4.2.5.4 Anforderungen GETPASS

Requirement Nr.	R-NS-007
Titel	GETPASS
Beschreibung	<p>Administratoren sollen das Passwort eines Benutzers einsehen können.</p> <p>Die Befehlssyntax wird wie folgt definiert: GETPASS <Nickname></p>
Kommentare	<p>Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben</p>

Tabelle 18: R-NS-007: GETPASS

4.2.5.5 Anforderungen GHOST

Requirement Nr.	R-NS-008
Titel	GHOST
Beschreibung	<p>Verliert ein Benutzer die Verbindung kann es vorkommen, das ein Abbild der Verbindung immer noch auf dem Server ist. Diese Verbindung soll getrennt werden können.</p>

	Die Befehlssyntax wird wie folgt definiert: GHOST <Nickname> <Passwort>
Kommentare	Benutzer muss für den Nickname identifiziert sein.

Tabelle 19: R-NS-008: GHOST

4.2.5.6 Anforderungen IDENTIFY

Requirement Nr.	R-NS-009
Titel	IDENTIFY
Beschreibung	Ein Benutzer muss sich für seinen Nickname identifizieren können. Wenn ein Benutzer das Passwort dreimal falsch eingibt, soll eine vorkonfigurierte Aktion ausgeführt werden. Die Befehlssyntax wird wie folgt definiert: IDENTIFY<Nickname> <Passwort>
Kommentare	Diese Funktion kann von jedem Benutzer verwendet werden.

Tabelle 20: R-NS-009: IDENTIFY

4.2.5.7 Anforderungen INFO

Requirement Nr.	R-NS-010
Titel	INFO
Beschreibung	Es sollen grundlegende Informationen über einen Nickname angezeigt werden können. Die Befehlssyntax wird wie folgt definiert: INFO <Nickname>
Kommentare	Diese Funktion kann von jedem Benutzer verwendet werden.

Tabelle 21: R-NS-010: INFO

4.2.5.8 Anforderungen LIST

Requirement Nr.	R-NS-011
Titel	LIST
Beschreibung	Die registrierten Nicknamen sollen in einer Liste angezeigt werden können. Es soll eine Suchmaske als Argument übergeben werden können. Die Befehlssyntax wird wie folgt definiert: LIST <Suchmaske>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 22: R-NS-011: LIST

4.2.5.9 Anforderungen LISTCHANS

Requirement Nr.	R-NS-012
Titel	LISTCHANS

Beschreibung	<p>Der Benutzer soll die Möglichkeit haben, alle Channels anzuzeigen in denen er erweiterte Rechte hat. Anzuzeigen sind Channel, Zugriff, wer den Zugriff hinzugefügt hat und das Datum.</p> <p>Die Befehlssyntax wird wie folgt definiert: LISTCHANS <Nickname></p>
Kommentare	<p>Um die Funktion zu verwenden, muss der Benutzer für den Nickname identifiziert sein.</p> <p>Benutzer mit Vollzugriff können die Liste für jeden Nickname anzeigen.</p>

Tabelle 23: R-NS-012: LISTCHANS

4.2.5.10 Anforderungen NOTIFY

Requirement Nr.	R-NS-013
Titel	NOTIFY
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Freunde verwaltet werden können. Wenn ein Freund sich zum Server verbindet sendet Nickserv eine Nachricht an den Benutzer.</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Befehlssyntax wird wie folgt definiert: NOTIFY [Befehl] < Nickname></p>
Kommentare	Der Benutzer muss für den Nickname identifiziert sein

Tabelle 24: R-NS-013: NOTIFY

4.2.5.11 Anforderungen REGISTER

Requirement Nr.	R-NS-014
Titel	REGISTER
Beschreibung	<p>Um einen Nickname zu reservieren und vor unbefugtem Zugriff zu schützen soll dieser registriert werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert: REGISTER <Passwort> <E-Mail Adresse></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 25: R-NS-014: REGISTER

4.2.5.12 Anforderungen RELEASE

Requirement Nr.	R-NS-015
Titel	RELEASE
Beschreibung	<p>Wird ein Nickname gesperrt, so muss der rechtmässige Benutzer die Möglichkeit haben die Sperre wieder aufzuheben.</p> <p>Die Befehlssyntax wird wie folgt definiert: RELEASE <Nickname> <Passwort></p>

Kommentare	
------------	--

Tabelle 26: R-NS-015: RELEASE

4.2.5.13 Anforderungen SET

Requirement Nr.	R-NS-015
Titel	SET
Beschreibung	<p>Der Benutzer soll die Möglichkeit haben, Einstellungen vorzunehmen:</p> <p>AUTHORIZE – Stellt ein, ob und für welche Liste die explizite Zustimmung gegeben werden muss.</p> <p>EMAIL – Legt die E-Mail Adresse fest</p> <p>HIDEEMAIL – E-Mail Adresse soll in der Nick Info angezeigt werden</p> <p>MFORWARD – Legt eine automatische Weiterleitung von Memos fest</p> <p>MLOCK – Legt Modi fest, die nach der Identifikation gesetzt werden.</p> <p>NOMEMO – Legt fest, ob ein Benutzer Memos erhalten will</p> <p>NOOP – Legt fest, ob der Benutzer in Channel in denen er Operatoren-Rechte hat, automatisch diesen Status erhalten soll.</p> <p>PASSWORD – Legt ein neues Passwort für den Nickname fest.</p> <p>PROTECT – Legt den Schutz für den Nickname fest. Es stehen drei Schutz-Stufen zur Verfügung</p> <p>URL: URL, die in der Nick Information angezeigt werden soll.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SET [Befehl] <Einstellung></p>
Kommentare	Der Benutzer muss für den Nickname identifiziert sein

Tabelle 27: R-NS-016: SET

4.2.5.14 Anforderungen SETPASS

Requirement Nr.	R-NS-016
Titel	SETPASS
Beschreibung	<p>Administratoren sollen das Passwort eines Benutzers neu setzen können</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SETPASS <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 28: R-NS-016: SETPASS

4.2.6 Weitere Nickserv Anforderungen

4.2.6.1 Identifikationstimer

Requirement Nr.	R-NS-017
Titel	Identifikationstimer
Beschreibung	Wird der Schutz eines Nicknames auf Normal gestellt, soll der Benutzer 60 Sekunden Zeit haben, sich zu identifizieren. Nach 30 Sekunden soll eine entsprechende Warnung ausgegeben werden.
Kommentare	

Tabelle 29: R-NS-017: Identifikationstimer

4.3 Chanserv

4.3.1 Use Case Chanserv (UC CS-01)

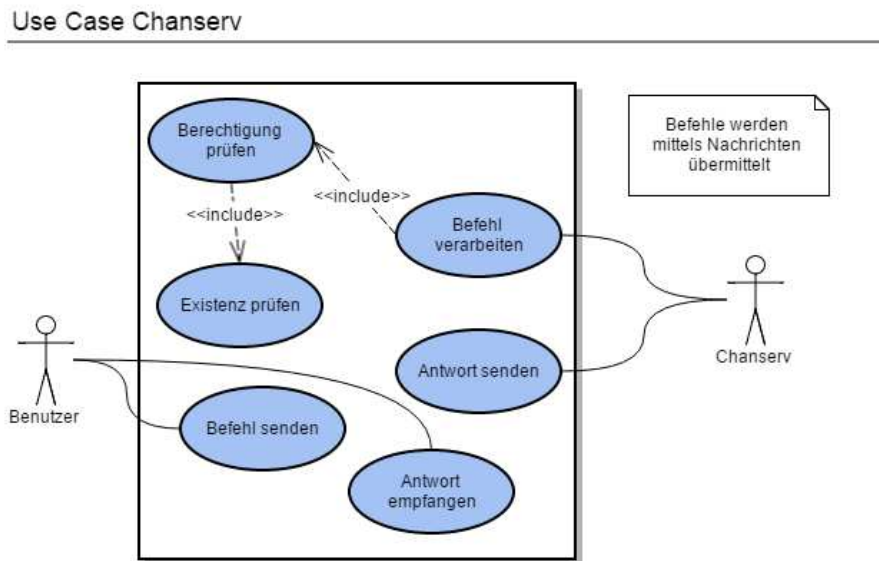


Abb. 8: Use Case Chanserv

4.3.2 Anforderungen Chanserv

Folgende Funktionen soll Chanserv anbieten

- **ACC** – Feststellung mit welcher Berechtigung ein Benutzer einen Nickname verwendet.
- **AKICK** – Eine Liste von Benutzeradressen, die automatisch aus dem Channel „gekickt“ werden.
- **AOP** – Steht für Auto Operator. Benutzer in dieser Liste haben Operatoren-Berechtigungen für den Channel.
- **DEHALFOP** – Entfernt den Half-Operatoren Status eines Benutzers
- **DEVOICE** – Entfernt den Voice Status eines Benutzers
- **DROP** – Löscht einen Channel
- **GETPASS** – Zeigt das Passwort des Channels an.
- **HALFOP** – Gibt einem Benutzer Half-Operatoren Status
- **HOP** – Steht für Half-Operator. Ein Half-Operator hat eingeschränkte Operatoren-Rechte.
- **IDENTIFY** – Identifikation für einen Channel
- **INFO** – Zeigt Information über einen Channel an
- **INVITE** – Ermöglicht es einem Benutzer der mindestens User Operator (Uop) Recht in einem Channel hat, sich selber in einen Channel einzuladen, falls nur eingeladene Benutzer den Channel betreten dürfen.
- **LIST** – Zeigt eine Liste von registrierten Channel an.
- **MKICK** – Kickt alle Benutzer aus einem Channel.
- **MDEOP** – Entfernt den Operatorenstatus von allen Operatoren in einem Channel.
- **OP** – Gibt einem Benutzer den Operatorenstatus in einem Channel.
- **REGISTER** – Einen Channel registrieren.
- **SET** – Einstellungen für den Channel. Diese sind:
 - **BOT** – Legt einen existierenden Bot fest, der für den Channel zuständig ist und diesen offen hält.

- **DESC** – Legt die Beschreibung des Channel fest.
- **FOUNDER** – Legt den Gründer des Channels fest. Muss ein registrierter Nickname sein.
- **OPWATCH** – Legt einen höheren Schutz für den Channel fest. Es können Benutzer mit explizitem Zugriff (Aop oder höher) Operatoren-Rechte gegeben werden.
- **LEAVEOPS** – Legt einen tieferen Schutz für den Channel fest. Man kann grundsätzlich jedem Benutzer Operatoren-Rechte geben.
- **KEEPTOPIC** – Legt fest, ob das Topic (Titel des Channels) beim Schliessen des Channels gemerkt und bei der erneuten Öffnung dessen neu gesetzt werden soll.
- **MEMOLEVEL** – Legt den Level fest, den ein Benutzer haben muss, um ein Memo an alle Benutzer mit Zugriff auf den Channel zu senden (Memoserv noch nicht implementiert)
- **MLOCK** – Legt Channelmodi fest, die nicht geändert werden können.
- **PASSWORD** – Legt ein neues Passwort für den Channel fest.
- **RESTRICTED** – Legt fest, ob nur Benutzer mit explizitem Zugriff (Uop oder höher) den Channel betreten dürfen.
- **SUCCESSOR** – Legt einen Nachfolger für den Channel fest. Wird automatisch als Founder gesetzt wenn der Nickname des Founders gelöscht wird.
- **TOPICLOCK** – Legt den Level fest, den ein Benutzer haben muss um das Topic des Channel zu ändern.
- **SETPASS** – Legt ein neues Passwort für einen Channel fest.
- **SOP** – Steht für Super Operator. Benutzer in dieser Liste haben Operatoren-Berechtigungen für den Channel und können andere Benutzer der Aop Liste (oder tiefer) hinzufügen.
- **UNBAN** – Löscht einen Ban, der auf einen Benutzer zutrifft, der Zugriff auf den Channel hat.
- **UOP** – Steht für User Operator. Benutzer in dieser Liste haben keinerlei Rechte, ausser, dass sie als Benutzer mit Zugriff auf den Channel gelten.
- **VOICE** – Gibt einem Benutzer Voice Status in einem Channel.
- **VOP** – Steht für Voice Operator. Benutzer in dieser Liste haben Voice-Berechtigungen für den Channel. Das heisst, dass sie in einem moderierten Channel (Channel Modus „voice only“ Nachrichten an den Channel senden können.

4.3.3 Anforderung Chanserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Chanserv sind identisch mit jenen von Nickserv.

4.3.4 Konfiguration Chanserv

Requirement Nr.	R-CS-001
Titel	Konfiguration Chanserv
Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Chanserv anbieten:</p> <p>Abschnitt „general“ enabled: Chanserv aktivieren name: Nickname von Chanserv realname: Realname von Chanserv</p> <p>Abschnitt „settings“: Welchen Zugriffslevel ein Benutzer für die Einstellungen der folgenden Einstellungen benötigt. Die Level sind wie folgt definiert: 6: Successor: Nickname hat Successor Level im Channel</p>

	<p>7: Der Benutzer hat sich für den Founder (Gründer) Nickname identifiziert.</p> <p>8: Der Benutzer hat sich für über Chanserv für den Channel identifiziert.</p> <p>Abschnitt „access“ Dieser Abschnitt beschreibt den benötigten Zugriffslevel um den entsprechenden Befehl auszuführen. Diese sind: 1 Uop 2 Vop 3 Hop 4 Aop 5 Sop 6 Successor 7 Für den Founder Nickname identifiziert 8 Über Chanserv für den Channel identifiziert. Folgende Unterabschnitte und Befehle müssen konfigurierbar sein:</p> <ul style="list-style-type: none"> • Abschnitt akick mit den Befehlen add, del, list, wipe • Abschnitt sop mit den Befehlen add, del, list, wipe • Abschnitt aop mit den Befehlen add, del, list, wipe • Abschnitt hop mit den Befehlen add, del, list, wipe • Abschnitt vop mit den Befehlen add, del, list, wipe • Abschnitt uop mit den Befehlen add, del, list, wipe <p>Sowie die Befehle mkick und mdeop.</p> <p>Bei den Zugriffskonfigurationen muss dabei eine Validierung vorgenommen werden, dass die Zugriffe Sinn ergeben. Das heisst zum Beispiel, dass ein Aop keine Sop hinzufügen können darf.</p> <p>Abschnitt „list“ maxlist: Max. Anzahl Einträge die beim LIST Befehl angezeigt werden soll operonly: Nur IRC Operatoren können diesen Befehl nutzen</p> <p>Abschnitt „registration“ delay: Wieviele Sekunden vergangen sein müssen, bevor erneut ein Nickname registriert werden kann</p> <p>Abschnitt „password“ getpass: Benötigter Level um diesen Befehl zu nutzen setpass: Benötigter Level um diesen Befehl zu nutzen</p> <p>Abschnitt „default“ Legt die Standardeinstellungen fest, die bei der Registrierung eines Channels gesetzt werden sollen:</p> <p>keeptopic: Topic soll beim Schliessen des Channel gemerkt werden leaveops: Operatoren ohne Zugriff sollen erlaubt sein</p>
--	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

	memolevel: Benötigter Zugriffslevel um Memos an Benutzer des Channels zu senden opwatch: Operatoren ohne Zugriff sollen nicht erlaubt sein restricted: Nur Benutzer mit Zugriff sollen den Channel betreten können topiclock: Benötigter Zugriffslevel um das Topic des Channels zu ändern autovop: Benutzer mit Vop Zugriff erhalten automatisch den Voice Status mlock: Channelmodi die nicht geändert werden können.
Kommentare	

Tabelle 30: R-CS-002: Konfiguration Chanserv

4.3.5 Anforderungen Chanserv Unterfunktionen

4.3.5.1 ACC

Requirement Nr.	R-CS-002
Titel	ACC
Beschreibung	<p>Es soll ein Befehles namens ACC implementiert der dem Absender die Berechtigung, die der angegebene Nutzer für den angegebenen Channel hat, zurückgibt.</p> <p>Folgendes soll zurückgegeben werden:</p> <p>Zugriffslevel Wieso der Benutzer diesen Zugriff hat.</p> <p>Beispiel Benutzer B hat Aop Zugriff auf Channel C, weil er sich für den Nickname A identifiziert hat</p> <p>Die Befehlssyntax wird wie folgt definiert: ACC <Channel> <Nickname></p>
Kommentare	Mindestens Uop Zugriff ist erforderlich.

Tabelle 31: R-CS-002: ACC

4.3.5.2 Anforderungen AKICK

Requirement Nr.	R-CS-003
Titel	AKICK
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Einträge von Benutzeradressen in einem Channel verwaltet werden können, für die Benutzer zu denen diese Adresse gehören automatisch aus dem Channel gekickt werden.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Die Maske muss validiert werden und folgendes Format aufweisen:</p> <p>Nickname!Benutzer@host.tld</p>

	<p>Sterne als Platzhalter werden akzeptiert. Beispiele:</p> <p>Nick!user@host *!*@host</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AKICK <Channel> [Befehl] <Adresse ></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 32: R-CS-003: AKICK

4.3.5.3 Anforderungen AOP

Requirement Nr.	R-CS-004
Titel	AOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Auto Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Operatoren-Rechte geben und die konfigurierten Befehle ausführen.</p> <p>Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 33: R-CS-004: AOP

4.3.5.4 Anforderungen DE-/HALFOP

Requirement Nr.	R-CS-005
Titel	HALFOP / DEHALFOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein Nickname, der Hop Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>HALFOP <Channel> <Nickname> DEHALFOP <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 34: R-CS-005: DE-/HALFOP

4.3.5.5 Anforderungen DE-/OP

Requirement Nr.	R-CS-006
Titel	DE-/OP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein Nickname, der Operatoren Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>OP <Channel> <Nickname> DEOP <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 35: R-CS-006: DE-/OP

4.3.5.6 Anforderungen DE-/VOICE

Requirement Nr.	R-CS-007
Titel	DE-/VOICE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein Nickname, der Vop Zugriff auf einen Channel hat sich oder andere Nutzer den Status über Chanserv gewähren oder entziehen kann. Nutzer ohne Zugriff sollen einen entsprechenden Hinweis ausgegeben erhalten.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOICE <Channel> <Nickname> DEVOICE <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 36: R-CS-007: DEVOICE

4.3.5.7 Anforderungen DROP

Requirement Nr.	R-CS-008
Titel	DROP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit der ein registrierter Channel gelöscht werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>DROP <Channel></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 37: R-CS-008: DROP

4.3.5.8 Anforderungen GETPASS

Requirement Nr.	R-CS-009
Titel	GETPASS

Beschreibung	Administratoren sollen das Passwort eines Channel einsehen können. Wird dieser Befehl benutzt muss ein Logeintrag geschrieben werden. Die Befehlssyntax wird wie folgt definiert: GETPASS <Channel>
Kommentare	Der Benutzer benötigt die konfigurierten Zugriffsrechte für diesen Befehl.

Tabelle 38: R-CS-009: GETPASS

4.3.5.9 Anforderungen HOP

Requirement Nr.	R-CS-010
Titel	HOP
Beschreibung	Es soll eine Liste geschaffen werden, mit der Half Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Half-Operatoren-Rechte geben und die konfigurierten Befehle ausführen. Es sollen folgende Unterfunktionen geschaffen werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen. Die Befehlssyntax wird wie folgt definiert: HOP <Channel> [Befehl] <Nickname>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 39: R-CS-010: HOP

4.3.5.10 Anforderungen IDENTIFY

Requirement Nr.	R-CS-011
Titel	IDENTIFY
Beschreibung	Es soll ein Mechanismus geschaffen werden, mit der ein Benutzer sich für den Channel identifizieren kann. Wenn der Benutzer sich identifiziert hat, erhält er den Vollzugriff für den Channel. Die Befehlssyntax wird wie folgt definiert: chanserv IDENTIFY<Channel> <Passwort>
Kommentare	

Tabelle 40: R-CS-011: IDENTIFY

4.3.5.11 Anforderungen INVITE

Requirement Nr.	R-CS-012
Titel	INVITE

Beschreibung	<p>Da es möglich ist, dass Channels nur Einladung betreten werden können muss ein Befehl geschaffen werden, mit der Benutzer die Zugriff auf den Channel haben sich über Chanserv selber einladen können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>INVITE <Channel></p>
Kommentare	Mindestens Uop Zugriff benötigt.

Tabelle 41: R-CS-013: INVITE

4.3.5.12 Anforderungen LIST

Requirement Nr.	R-CS-013
Titel	LIST
Beschreibung	<p>Die registrierten Channels sollen in einer Liste angezeigt werden können. Es soll eine Suchmaske als Argument übergeben werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LIST <Suchmaske></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 42: R-CS-013: LIST

4.3.5.13 Anforderungen MDEOP

Requirement Nr.	R-CS-014
Titel	MDEOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit welchem sämtlichen Operatoren eines Channel der Status entzogen werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MDEOP <Channel></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 43: R-CS-014: MDEOP

4.3.5.14 Anforderungen MKICK

Requirement Nr.	R-CS-015
Titel	MKICK
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit welchem sämtliche Benutzer aus einem Channel gekickt werden können. Wird dieser Befehl ausgeführt soll Chanserv automatisch den Channel vorübergehend betreten damit dieser offen bleibt.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MKICK <Channel> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für diesen Befehl haben

Tabelle 44: R-CS-015: MKICK

4.3.5.15 Anforderungen REGISTER

Requirement Nr.	R-CS-016
Titel	REGISTER
Beschreibung	<p>Soll ein Befehl geschaffen werden, mit dem man einen Channel mit Chan-serv registrieren kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>REGISTER <Channel> <Passwort> <Beschreibung></p>
Kommentare	<p>Der Nutzer muss einen registrierten Nickname haben.</p> <p>Der Nutzer muss die konfigurierten Rechte für die Registrierung von Channels haben</p>

Tabelle 45: R-CS-016: REGISTER

4.3.5.16 Anforderungen SET

Requirement Nr.	R-CS-017
Titel	SET
Beschreibung	<p>Der Gründer eines Channels soll die Möglichkeit haben, Einstellungen vorzunehmen:</p> <p>BOT – Legt einen Bot für den Channel fest. Der Bot muss über Botserv registriert werden.</p> <p>DESCRIPTION – Legt die Beschreibung des Channels fest</p> <p>FOUNDER – Legt einen neuen Gründer für den Channel fest. Der Gründer muss einen registrierten Nickname haben.</p> <p>KEEPTOPIC – Legt fest, ob das Topic beim Schliessen des Channel gemerkt werden soll.</p> <p>LEAVEOPS – Legt fest, ob Benutzer ohne Aop Zugriff oder höher Op-Rechte erhalten können. Wenn diese Option aktiviert wird soll Opwatch automatisch deaktiviert werden.</p> <p>MEMOLEVEL – Legt fest, welchen Zugriff ein Benutzer auf den Channel haben muss, um Memos an alle Benutzer mit Zugriff zu schicken.</p> <p>MLOCK – Legt Modi fest, die nicht verändert werden können.</p> <p>OPWATCH – Legt fest, ob nur Benutzer mit entsprechender Berechtigung Op Rechte erhalten kann. Wenn diese aktiviert wird muss Leaveops automatisch deaktiviert werden.</p> <p>PASSWORD – Legt ein neues Passwort für den Nickname fest.</p> <p>RESTRICTED – Nur Benutzer mit Uop Zugriff oder höher dürfen den Channel betreten.</p> <p>SUCCESSOR – Legt einen Nachfolger für den Channel fest. Wenn der Nickname des Founders gelöscht wird soll automatisch der Nachfolger als Founder gesetzt werden. Der Nachfolger muss einen registrierten Nickname haben</p> <p>TOPICLOCK – Legt fest, welchen Zugriff ein Benutzer auf den Channel haben muss, um das Topic des Channels zu setzen. Mit OFF soll jeder Op das Topic ändern können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SET <Channel> [Befehl] <Einstellung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte haben.

Tabelle 46: R-CS-017: SET

4.3.5.17 Anforderungen SETPASS

Requirement Nr.	R-CS-018
Titel	SETPASS
Beschreibung	Administratoren sollen das Passwort eines Channel neu setzen können. Wird dieser Befehl benutzt muss ein Logeintrag geschrieben werden. Die Befehlssyntax wird wie folgt definiert: SETPASS <Channel> <Passwort>
Kommentare	Der Benutzer benötigt die konfigurierten Zugriffsrechte für diesen Befehl.

Tabelle 47: R-CS-018: SETPASS

4.3.5.18 Anforderungen SOP

Requirement Nr.	R-CS-019
Titel	SOP
Beschreibung	Es soll eine Liste geschaffen werden, mit der Super Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich über Chanserv Operatoren-Rechte geben und die konfigurierten Befehle ausführen. Es sollen folgende Unterfunktionen geschaffen werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste Es dürfen nur registrierte Nicknames der verwendet werden. Dies zu prüfen. Die Befehlssyntax wird wie folgt definiert: SOP <Channel> [Befehl] <Nickname>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 48: R-CS-019: SOP

4.3.5.19 Anforderungen UNBAN

Requirement Nr.	R-CS-020
Titel	UNBAN
Beschreibung	Es soll ein Befehl geschaffen werden mit dem Benutzer die Uop Zugriff oder höher auf einen Channel haben, allfällige Bans aus dem Channel entfernen können. Alle Bans die auf den Benutzer zutreffen sollen gefunden und gelöscht werden. Die Befehlssyntax wird wie folgt definiert: UNBAN<Channel>
Kommentare	Der Benutzer benötigt Uop Zugriff oder höher

Tabelle 49: R-CS-020: UNBAN

4.3.5.20 Anforderungen UOP

Requirement Nr.	R-CS-021
Titel	UOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der User Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff gelten als Benutzer mit Zugriff auf den Channel, haben aber sonst keine speziellen Rechte. Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>UOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 50: R-CS-021: UOP

4.3.5.21 Anforderungen VOP

Requirement Nr.	R-CS-022
Titel	UOP
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit der Voice Operator Einträge verwaltet werden können. Benutzer mit diesem Zugriff können sich mit dem VOICE Befehl selber den Status im Channel geben. Es sollen folgende Unterfunktionen geschaffen werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an WIPE – Entfernt alle Einträge aus der Liste</p> <p>Es dürfen nur registrierte Nicknames verwendet werden. Dies zu prüfen.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOP <Channel> [Befehl] <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 51: R-CS-022: Uop

4.4 Operserv

4.4.1 Use Case Operserv (UC OS-01)

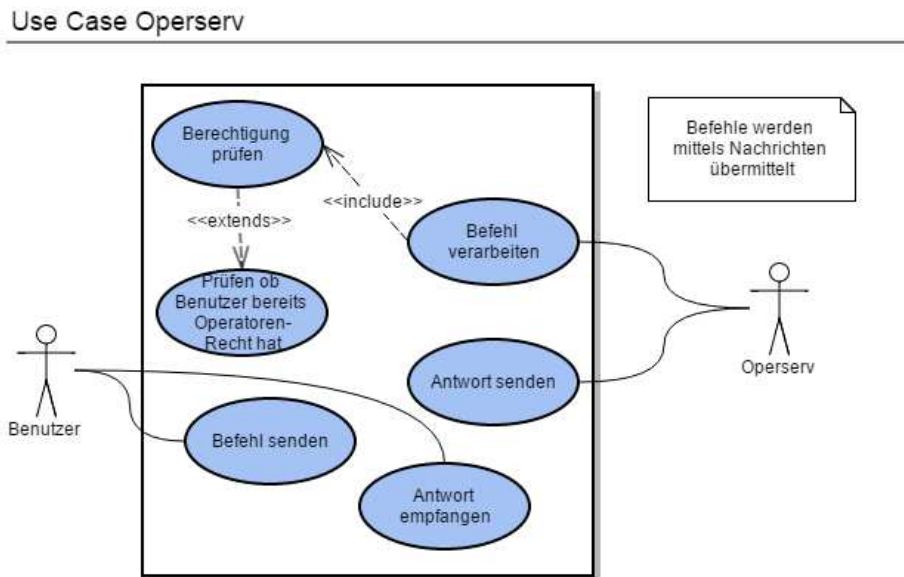


Abb. 9: Use Case Operserv

4.4.2 Anforderungen Operserv

Folgende Funktionen soll Operserv anbieten

- **AKILL** – Hinzufügen eines AUTOKILLS
- **CHATOPS** – Nachricht an alle Server Operatoren
- **CHGHOST** – Ändern des Hostnamen eines Benutzers
- **GLOBAL**–Nachricht an alle Server Operatoren auf dem Netzwerk
- **KILL**– Trennen der Verbindung eines Benutzers
- **LOCAL**–Nachricht an alle Server Operatoren auf dem Server, jedoch nicht in gesamten Netzwerk
- **OPER**– Verwalten der Operatoren-Liste
- **SGLINE** – Service G:Line. Dies bedeutet einen Ban auf dem gesamtem Netzwerk auf Basis der Hostmaske
- **SKLINE**– Service K:Line. Dies bedeutet einen Ban auf dem Lokalen Server auf Basis der Hostmaske
- **SQLINE** – Service Q:Line. Dies verhindert die Benutzung eines spezifischen Nickname
- **SZLINE**– Service Z:Line. Dies bedeutet einen Ban auf dem gesamten Netzwerk auf Basis der IP-Adresse. Der Unterschied zur G:Line ist dabei, dass ganze IP-Adressen Bereich gesperrt werden können.

4.4.3 Anforderung Operserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Operserv sind identisch mit jenen von Nickserv.

4.4.4 Konfiguration Operserv

Requirement Nr.	R-OS-001
Titel	Konfiguration Operserv
Beschreibung	Folgende Konfigurationsmöglichkeiten soll Operserv anbieten:

	<p>Abschnitt „general“ enabled: Opserv aktivieren name: Nickname von Opserv realname: Realname von Opserv access_flag: Welche Operatoren Zugriff ist nötig um Vollzugriff auf Opserv zu haben.</p> <p>Abschnitt „global“: Legt fest, bei welchen Events ein GLOBAL herausgeschickt werden soll, jeweils 0 oder 1 ist zulässig. Folgende Events sind konfigurierbar: on_oper, on_akill, on_sgline, on_skline, on_sqline, on_szline, on_kill</p> <p>Abschnitt „default“ Legt die Standard-Berechtigungen für einen neuen Oper fest. Jeweils 0 oder sind zulässig. Jeder Opserv Befehl ausser Oper ist zu berücksichtigen. Zusätzlich soll ein Standard Vhost (Virtual Host) gesetzt werden. Benutzer ohne Server Operatoren-Rechte aber mit Zugriff auf Opserv soll diesen erhalten.</p>
Kommentare	

Tabelle 52: R-OS-001: Konfiguration Opserv

4.4.5 Anforderungen Opserv Unterfunktionen

4.4.5.1 Anforderungen AKILL

Requirement Nr.	R-OS-002
Titel	AKILL
Beschreibung	<p>Es soll eine Liste geschaffen werden, mit dem die Verbindung von störenden Benutzern automatisch getrennt werden kann. Dies kommt einem Ban auf den Server gleich.</p> <p>Folgende Unterbefehle sollen unterstützt werden: ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an</p> <p>Als Zeitdauer sollen folgende Kürzel unterstützt werden:</p> <ul style="list-style-type: none"> • h – Stunden • w – Wochen • m – Monate • y – Jahre <p>Wird kein Kürzel angegeben, so ist die Zeitdauer in Minuten zu interpretieren. Die Dauer soll optional sein. Wird keine festgelegt so ist der Auto-kill permanent bis er manuell entfernt wird.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>AKILL [Befehl] <Adresse > <Dauer> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 53: R-OS-002: AKILL

4.4.5.2 Anforderungen CHATOPS

Requirement Nr.	R-OS-003
Titel	CHATOPS
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine CHATOPS Nachricht an alle Operatoren gesendet werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>CHATOPS <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 54: R-OS-003: CHATOPS

4.4.5.3 Anforderungen CHGHOST

Requirement Nr.	R-OS-004
Titel	CHGHOST
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein CHGHOST Befehl abgesetzt werden kann. Dies soll den Host eines Benutzers ändern.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>CHGHOST <Benutzer> <Neuer Hostname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 55: R-OS-004: CHGHOST

4.4.5.4 Anforderungen GLOBAL

Requirement Nr.	R-OS-005
Titel	GLOBAL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein GLOBAL Befehl abgesetzt werden kann. So kann man eine Nachricht an alle Server Operatoren auf dem gesamten Netzwerk senden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>GLOBAL <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 56: R-OS-005: GLOBAL

4.4.5.5 Anforderungen KILL

Requirement Nr.	R-OS-006
Titel	KILL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem die Verbindung eines störenden Benutzers getrennt werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>KILL <Benutzer> <Begründung></p>

Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben
------------	------------------------------------------------------------------------------------

Tabelle 57: R-OS-006: KILL

4.4.5.6 Anforderungen LOCAL

Requirement Nr.	R-OS-007
Titel	GLOBAL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein LOCOPS Befehl abgesetzt werden kann. So kann man eine Nachricht an alle Server Operatoren auf dem aktuellen Server senden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LOCAL <Nachricht></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 58: R-OS-007: LOCAL

4.4.5.7 Anforderungen OPER

Requirement Nr.	R-OS-008
Titel	OPER
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem Benutzer, die noch über keine Operatoren-Rechte verfügen, solche gewähren kann. Der Zugriff auf die einzelnen Befehle soll individuell eingestellt werden können.</p> <p>Folgende Unterbefehle sollen unterstützt werden:</p> <p>ADD – Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste LIST – Zeigt alle Einträge der Liste an SET – Legt die individuellen Einstellungen fest. Folgende Rechte können eingestellt werden:</p> <p>AKILL CHGHOST CHATOPS GLOBAL LOCAL KILL SGLINE SKLINE SQLINE SZLINE VHOST</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>OPER [Befehl] <Nickname> <Zugriff> [ENABLE DISABLE <Hostname>]</p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 59: R-OS-008: OPER

4.4.5.8 Anforderungen Server-Bans und Nicksperrn

Requirement Nr.	R-OS-009
Titel	SGLINE / SKLINE/ SZLINE / SQLINE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine neue G:Line, K:Line, Z:Line oder Q:Line gesetzt werden kann.</p> <p>Folgende Unterbefehle sollen unterstützt werden: ADD –Fügt der Liste einen Eintrag hinzu DEL – Entfernt einen Eintrag aus der Liste</p> <p>Als Zeitdauer sollen folgende Kürzel unterstützt werden:</p> <ul style="list-style-type: none"> • h – Stunden • w – Wochen • m – Monate • y – Jahre <p>Wird kein Kürzel angegeben, so ist die Zeitdauer in Minuten zu interpretieren. Die Dauer soll optional sein. Wird keine festgelegt so ist die G:LINE permanent bis sie manuell entfernt wird.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SGLINE [Befehl] <Adresse > <Dauer> <Begründung> SKLINE [Befehl] <Adresse > <Dauer> <Begründung> SZLINE [Befehl] <Adresse > <Dauer> <Begründung> SQLINE [Befehl] <Nickname> <Dauer> <Begründung></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 60: R-OS-009: SGLINE / SKLINE/ SZLINE / SQLINE

4.5 Botserv

4.5.1 Use Case Botserv (UC BS-01)

Use Case Botserv

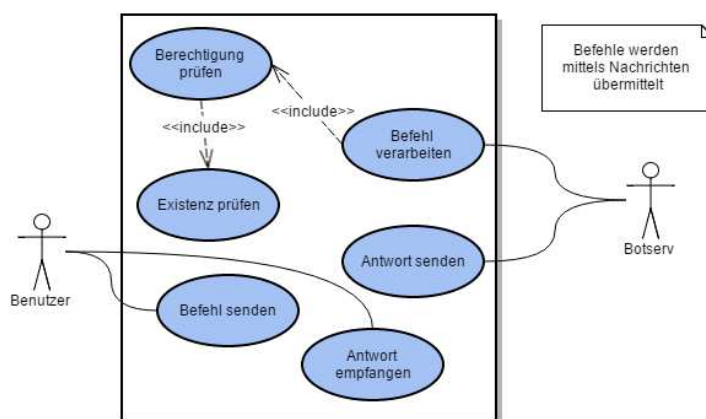


Abb. 10: Use Case Botserv

4.5.2 Anforderungen Botserv

Folgende Funktionen soll Botserv anbieten

- **ADD** – Hinzufügen eines Bots
- **DEL**– Entfernen eines Bots
- **DEHALFOP**– Entziehen des H
- **DEOP** – Entziehen des
- **DEVOICE**– Entziehen des
- **GETPASS**– Einsehen des Passwortes eines Bots. s
- **HALFOP**– Gewähren des Half-Operator Status
- **IDENTIFY**– Identifikation für den Bot
- **INFO**–Informationen über einen Bot anzeigen
- **KICK**– Kicken eines Benutzers aus dem Channel
- **LIST**– Liste der Bots anzeigen lassen.
- **MSG**– Nachricht über den Bot an einen Channel senden
- **OP** – Gewähren des Operator Status
- **SET** – Einstellungen am Bot vornehmen. Folgende Einstellungen können vorgenommen werden:
 - **NAME**: Neuer Nickname für den Bot
 - **PASSWORD**: Neues Passwort
 - **REALNAME**: Neuer Realname
 - **USERNAME**: Neuer Username
- **SETPASS**: Neues Passwort für den Bot setzen.
- **VOICE**– Gewähren des Voice Operator Status

4.5.3 Anforderung Botserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Botserv sind identisch mit jenen von Nickserv.

4.5.4 Konfiguration Botserv

Requirement Nr.	R-BS-001
Titel	Konfiguration Botserv
Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Botserv anbieten:</p> <p>Abschnitt „general“ enabled: Botserv aktivieren name: Nickname von Botserv realname: Realname von Botserv</p> <p>Abschnitt „access“: Legt fest, welchen Operatoren Zugriff ein Benutzer haben muss, um den jeweiligen Befehl auszuführen. Folgende Werte sind zulässig: 0: Jeder Benutzer darf den Befehl verwenden 1: Help Operatoren 2: IRC Operatoren 3: Co Admins 4: Server Admins 5: Services Admins 6: Network Admins</p>

	Folgende Befehle sollen konfigurieren werden können: add, del, list, set, getpass, setpass
Kommentare	

Tabelle 61: R-OS-001: Konfiguration Botserv

4.5.5 Anforderungen Botserv Unterfunktionen

4.5.5.1 Anforderungen ADD/DEL

Requirement Nr.	R-BS-002
Titel	ADD/DEL
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein neuer Bot erstellt oder ein bestehender gelöscht werden kann. Beim Erstellen sollen der Nickname des Bots auch für den Username und den Realname gesetzt werden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>ADD <Name> <Passwort> DEL <Name></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 62: R-BS-002: ADD/DEL

4.5.5.2 Anforderungen DE-/HALFOP

Requirement Nr.	R-BS-003
Titel	DE-/HALFOP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Half-Operatoren Rechte geben oder entziehen kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>HALFOP <Bot> <Channel> <Nickname> DEHALFOP <Bot> <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 63: R-BS-003: DE-/HALFOP

4.5.5.3 Anforderungen DE-/VOICE

Requirement Nr.	R-BS-004
Titel	DE-/VOICE
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Voice-Operatoren Rechte geben oder entziehen kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>VOICE <Bot> <Channel> <Nickname> DEVOICE <Bot> <Channel> <Nickname></p>

Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben
------------	------------------------------------------------------------------------------------

Tabelle 64: R-BS-004: DE-/VOICE

4.5.5.4 Anforderungen DE-/OP

Requirement Nr.	R-BS-005
Titel	DE-/OP
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man einem Benutzer vom Bot im angegebenen Channel Operatoren Rechte geben oder entziehen kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>OP <Bot> <Channel> <Nickname> DEOP <Bot> <Channel> <Nickname></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 65: R-BS-005: DE-/OP

4.5.5.5 Anforderungen GETPASS

Requirement Nr.	R-BS-006
Titel	GETPASS
Beschreibung	<p>Administratoren sollen das Passwort eines Bots einsehen können. Wird dieser Befehl verwendet muss ein Log-Eintrag geschrieben werden.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>GETPASS<Bot></p>
Kommentare	Der Nutzer muss die konfigurierten Zugriffsrechte für die jeweiligen Befehle haben

Tabelle 66: R-BS-006: GETPASS

4.5.5.6 Anforderungen IDENTIFY

Requirement Nr.	R-BS-007
Titel	IDENTIFY
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem ein Benutzer sich für den Bot identifizieren kann und so den vollen Zugriff bekommt.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>IDENTIFY <Bot> <Passwort></p>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 67: R-BS-007: IDENTIFY

4.5.5.7 Anforderungen INFO

Requirement Nr.	R-BS-008
Titel	INFO
Beschreibung	Es soll ein Befehl geschaffen werden mit dem Informationen über einen Bot angezeigt werden können. Angezeigt werden sollen:

	<p>Username, Hostname, Realname und Channels auf dem der Bot ist.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>INFO <Bot></p>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 68: R-BS-008: INFO

4.5.5.8 Anforderungen KICK

Requirement Nr.	R-BS-009
Titel	KICK
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem störende Benutzer aus dem Channel gekickt werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>KICK <Bot> <Channel> <Nickname> <Begründung></p>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 69: R-BS-009: KICK

4.5.5.9 Anforderungen LIST

Requirement Nr.	R-BS-010
Titel	LIST
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem eine Liste aller existierenden Bots angezeigt werden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>LIST</p>
Kommentare	Dieser Befehl erfordert keine besonderen Rechte.

Tabelle 70: R-BS-010: LIST

4.5.5.10 Anforderungen KICK

Requirement Nr.	R-BS-011
Titel	MSG
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem Nachrichten mit dem Bot als Absender an den Channel gesendet werden können.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>MSG <Bot> <Channel> <Nachricht></p>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 71: R-BS-011: MSG

4.5.5.11 Anforderungen SET

Requirement Nr.	R-BS-012
Titel	SET

Beschreibung	Es soll ein Befehl geschaffen werden mit dem folgende Einstellungen für den Bot vorgenommen werden können: NAME, PASSWORD, REALNAME, USERNAME Die Befehlssyntax wird wie folgt definiert: SET <Bot> [Einstellung] <Wert>
Kommentare	Der Benutzer muss für den Bot identifiziert sein.

Tabelle 72: R-BS-012: SET

4.5.5.12 Anforderungen SETPASS

Requirement Nr.	R-BS-013
Titel	SETPASS
Beschreibung	Administratoren sollen das Passwort für einen Bot neu setzen können Die Befehlssyntax wird wie folgt definiert: SETPASS <Bot> <Passwort>
Kommentare	Der Benutzer muss die konfigurierten Rechte für diesen Befehl haben

Tabelle 73: R-BS-013: SETPASS

4.6 Adminserv

4.5.1 Use Case Adminserv (UC AS-01)

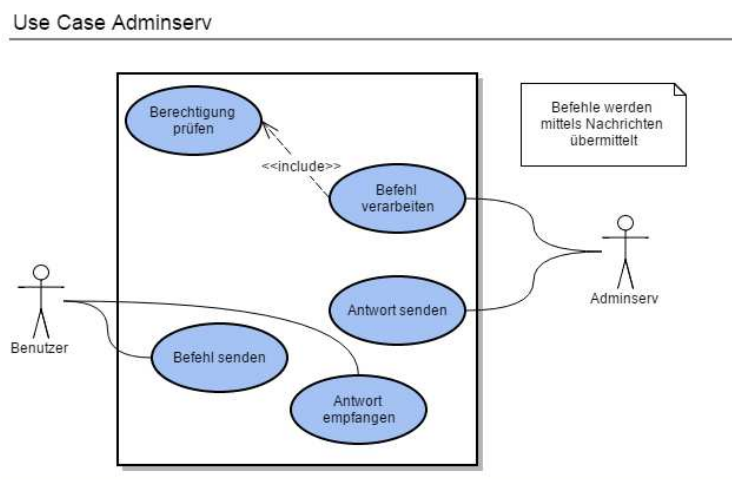


Abb. 11: Use Case Adminserv

4.6.2 Anforderungen Adminserv

Folgende Funktionen soll Adminserv anbieten

- **SAVEDATA** – Manuelles Speichern der Datenbank
- **SQUIT**– Beenden von Services

4.6.3 Anforderung Adminserv Hauptfunktion

Die Anforderungen für die Hauptfunktion von Adminserv sind identisch mit jenen von Nickserv.

4.6.4 Konfiguration Adminserv

Requirement Nr.	R-AS-001
Titel	Konfiguration Adminserv

Beschreibung	<p>Folgende Konfigurationsmöglichkeiten soll Botserv anbieten:</p> <p>Abschnitt „general“ enabled: Adminserv aktivieren name: Nickname von Adminserv realname: Realname von Adminserv</p>
Kommentare	

Tabelle 74: R-AS-001: Konfiguration Adminserv

4.6.5 Anforderungen Adminserv Unterfunktionen

4.6.5.1 Anforderungen SAVEDATA

Requirement Nr.	R-AS-002
Titel	SAVEDATA
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man die gesamte Datenbankbankbank manuell speichern kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SAVEDATA</p>
Kommentare	Der Benutzer muss Services Administrator Zugriff haben.

Tabelle 75: R-AS-002: SAVEDATA

4.6.5.2 Anforderungen SQUIT

Requirement Nr.	R-AS-003
Titel	SQUIT
Beschreibung	<p>Es soll ein Befehl geschaffen werden mit dem man die Services aus dem Chat beenden kann.</p> <p>Die Befehlssyntax wird wie folgt definiert:</p> <p>SQUIT</p>
Kommentare	Der Benutzer muss Services Administrator Zugriff haben.

5. Umsetzung

Nachdem wir nun alle Anforderungen dokumentiert haben müssen wir uns Gedanken über die Umsetzung machen. Dafür wollen wir uns zunächst überlegen, wie die Architektur aussehen soll. Danach überlegen wir uns, wie die Datenbank auszusehen hat und widmen uns dann der effektiven Umsetzung des Projekts.

5.1 Design

5.1.1 Architektur

Da die Dienste auf dem Basis-Server laufen und dieser wiederum eine Verbindung zum IRC Server herstellt werden die Befehle vom Client an den IRC Server gesendet. Dieser ist verantwortlich dafür, dass diese weitergeleitet werden. Da wir in den Anforderungen festgehalten haben, dass die Befehle als PRIVMSG an den jeweiligen Dienst gesendet werden müssen wir uns über die Implementierung keine Gedanken machen. Diese Art der Befehlsübermittlung bietet zudem den Vorteil, dass wir auf die Anwendung Threads verzichten können, denn der IRC-Server stellt hier eine Art „Flaschenhals“ dar da jeweils nur ein Befehl auf einmal abgearbeitet werden.

Die Daten werden jeweils beim Start in den Arbeitsspeicher geladen und sämtliche Änderungen werden im Arbeitsspeicher vorgenommen. In regelmässigen Abständen werden dann die Daten in die SQLite Datenbank geschrieben.

Das Architektur-Design sieht demnach wie folgt aus:

Architektur IRC Services

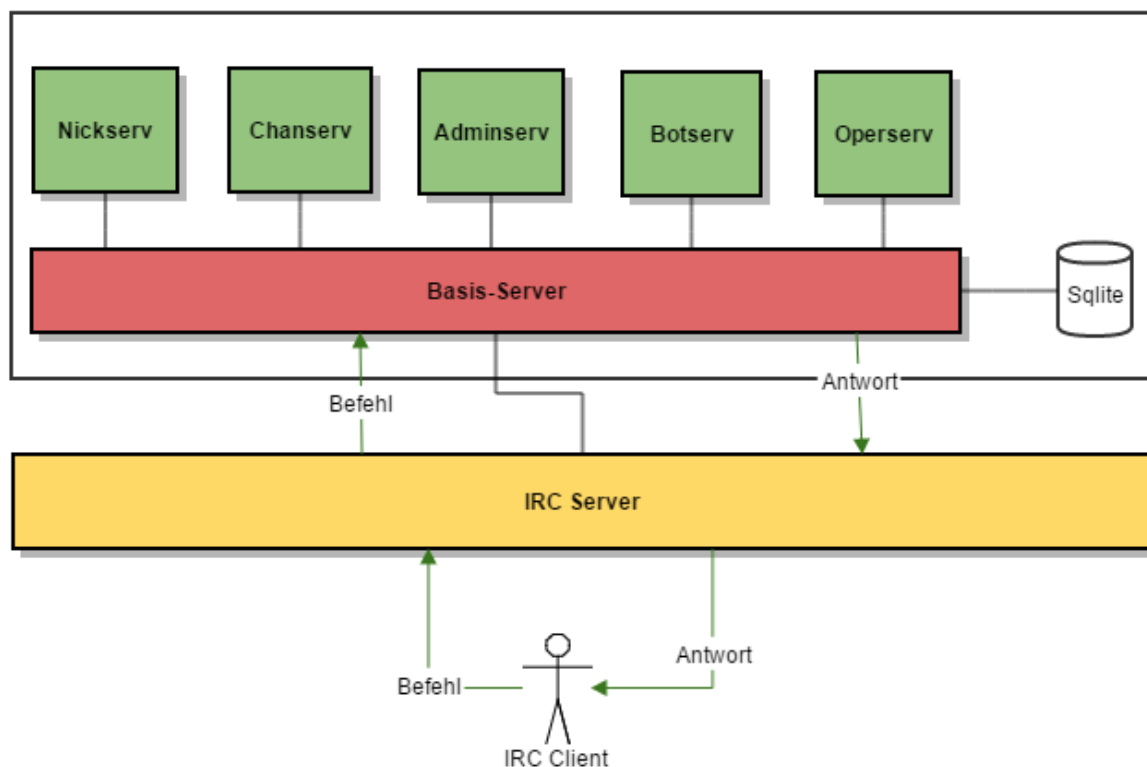


Abb. 12: Architektur

5.1.2 Datenbank-Design

Wie erwähnt werden wir als Datenbank SQLite verwenden. Das hat den Vorteil, dass die IRC Services keine weitere Software benötigt um lauffähig zu sein.

Folgendes Design wurde festgelegt:

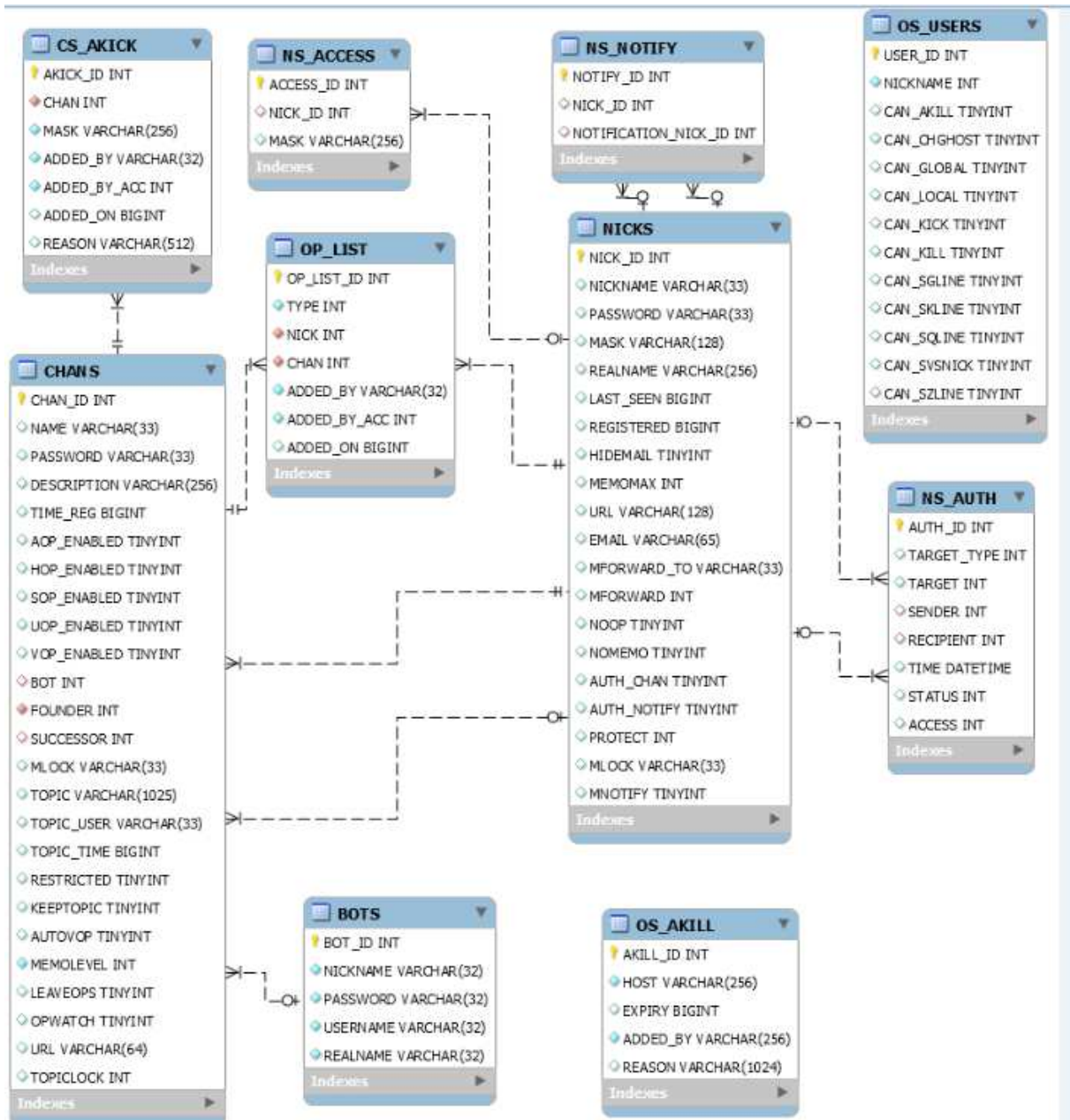


Abb. 13: Datenbank-Schema

5.2 Setup IRC Server

Der IRC Server muss zunächst so konfiguriert werden, dass er entsprechende Service-Verbindungen akzeptieren kann.

UnrealIRCd bietet als Konfiguration den Link-Block an⁵.

```
link <server-name> {
    username <usermask>;
    hostname <ipmask>;
    bind-ip <ip-to-bind-to>;
    port <port-to-connect-on>;
    password-connect <password-to-connect-with>;
    password-receive <password-to-receive> { <auth-type>; };
    hub <hub-mask>;
    leaf <leaf-mask>;
    leafdepth <depth>;
    class <class-name>;
    ciphers <ssl-ciphers>;
    options {
        <option>;
        <option>;
        ...
    };
};
```

Abb. 14: Konfiguration Unrealircd

Folgendes Beispiel stellt eine lauffähige Konfiguration dar:

```
link services.reefmaster.ch
{
    username      *;
    hostname      *;
    bind-ip       *;
    port          7029;
    hub           *;
    password-connect "pass123";
    password-receive "pass123";
    class         servers;
};
```

Abb. 15: Beispiel-Konfiguration Link

5.3. Implementierung Konfiguration

Da die Applikation für verschiedene Zwecke verwendet werden soll muss sie komplett konfigurierbar sein. Um eine Konfigurationsdatei validieren zu können gibt es verschiedene existierende Lösungen. Die Wahl fiel auf Confuse 2.5, weil die Verwendung relativ simpel ist. Confuse kommt als Bibliothek die im Quellcode eingebunden werden kann.

Die Konfigurationsdatei kann mit Confuse in Blöcke und Abschnitte unterteilt werden. Beispiel aus der Konfigurationsdatei:

```

services
{
    general
    {
        name = "services.reefmaster.ch"
        description = "Reefmaster IRC Services"
        user = "services"
        host = "reefmaster.ch"
        port = 7029
        address = "localhost"
        irc = "irc.reefmaster.ch"
    }
}

```

Abb. 16: Beispiel Basis-Server Konfiguration

Gleich nach dem Start der Applikation muss die Konfiguration geladen und validiert werden.

Als erstes wird die Konfiguration als normale Datei geöffnet. Dazu verwenden wir die `fopen` Funktion⁶:

```

int config_load(const char *file) {
    FILE *pFile;
    pFile = fopen(CONFIG_FILE, "r");
    if (!pFile) {
        printf(CONF_ERR_FILENOTFOUND"\n");
        addlog(2, CONF_LOG_ERR_FILENOT);
        return -1;
    }
}

```

Abb. 17: config_load

Anmerkung: den Namen der Konfigurationsdatei haben wir als literarische Konstante (`CONFIG_FILE`) definiert, damit wir im Falle von Änderungen diese nur an einem Ort vornehmen müssen. Diese stellen eine Art Platzhalter dar, die durch den Prozessor beim kompilieren entsprechend ersetzt werden⁷.

Als nächstes werden die einzelnen Blöcke definiert, die validiert werden sollen. Confuse stellt eine Datenstruktur zur Verfügung in die wir einzelne Blöcke deren Abschnitte direkt bereitstellen können. Diese wird beim kompilieren verarbeitet und wir sind in der Lage, jeden Abschnitt in sich zu validieren, indem eine sogenannte Callback-Funktion definiert werden kann. Beispiel:

```

static cfg_opt_t main_general_opts[] = {
    CFG_STR_CB("name", "services.mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("user", "user", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("host", "mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_INT_CB("port", 7029, CFGF_NONE, (void*)&config_port),
    CFG_STR_CB("description", "My Net", CFGF_NONE, (void*)&config_str128),
    CFG_STR_CB("help_channel", "#help", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("address", "localhost", CFGF_NONE, (void*)&config_str32),
    CFG_STR_CB("irc", "irc.mynet.org", CFGF_NONE, (void*)&config_str32),
    CFG_END()
};

```

Wir betrachten hier den Block „services“ mit dem Unterblock „general“. `cfg_opt_t` ist die Datenstruktur von Confuse. `CFG_STR_DB` ist ein Makro, also eine Art Script, die definiert wird um den Inhalt des

Makros beim kompilieren ausführt. Es verarbeitet Konfigurationsabschnitt und die angegebenen Daten in die Struktur ablegt. Folgende Parameter werden in diesem Makro erwartet:

- Name des Abschnitts
- Standard-Wert des Abschnitts, für den Fall, dass kein Wert übergeben wurde
- Spezielle Parameter (wird im gesamten Projekt nicht benötigt, daher immer CFGF_NONE)
- Zeiger auf die dazugehörige Callback-Funktion die den Abschnitt validieren soll.

Für jeden Hauptblock, wird zunächst der Unterblock mit seinen Abschnitt validiert. Danach werden alle Blöcke validiert:

```
static cfg_opt_t opts[] = {
    CFG_SEC("services", services_opts, CFGF_NONE),
    CFG_SEC("nickserv", nickserv_opts, CFGF_NONE),
    CFG_SEC("chanserv", chanserv_opts, CFGF_NONE),
    CFG_SEC("operserv", operserv_opts, CFGF_NONE),
    CFG_SEC("botserv", botserv_opts, CFGF_NONE),
    CFG_SEC("adminserv", adminserv_opts, CFGF_NONE),
    CFG_END()
};
```

Abb. 18: Validierung der Unterblöcke

Danach wird mit dem Befehl `cfg_parse` die gesamte Konfiguration geprüft und jede Callback Funktion ausgeführt. Ist etwas nicht sauber konfiguriert wird die Applikation nicht gestartet und wird stattdessen ein Fehler mit der Zeilennummer in der Konfiguration wo das Problem ist ausgegeben.

Treten keine Fehler auf, werden nun die Werte der Konfiguration den globalen Variablen, als den Variablen die in der gesamten Applikation verwendet werden, zugewiesen und die Applikation wird gestartet.

5.4. Implementierung Server-Verbindung

5.4.1 Verbindung Basis-Server

Um zu verstehen, wie ein Server sich zu einem anderen Server verbinden kann müssen wir uns zunächst über Sockets unterhalten. Ein Socket ist eine Abstraktion eines Kommunikations-Endpunktes⁸.

Ähnlich wie in einer Datei benutzen Applikationen Deskriptoren um über Socket zu kommunizieren. Um eine Netzwerk-Verbindung zu öffnen benötigen wir daher zuerst einen Socket mit seinen Einstellungen:

```
(unsigned char) dns->h_addr_list[0][3]
sock = socket(AF_INET, SOCK_STREAM, 0);
if (sock < 0) {
    printf(APP_ERR_SOCKET);
    return -1;
}
printf(APP_DBG_CONNECTINGTOSERVER, s unreal);
```

Abb. 19: Anlegen eines Sockets

Da das IRC Protokoll über TCP/IP Kommuniziert, verwenden wir die Option `AF_INET` (IPv4 Internet Domain). Danach müssen wir die Verbindung Konfigurieren. C stellt in der UNIX Umgebung die Struktur `sockaddr_in` zur Verfügung, die eine Adresse für einen Socket beschreiben. Damit können wir alles konfigurieren was nötig um die Verbindung herzustellen.

```
bzero(&addr, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons((unsigned short) port);
addr.sin_addr.s_addr = inet_addr(serverip);
```

Abb. 20: Konfiguration Socket und Adresse

Nun können wir mit `Connect` die Verbindung herstellen:

```
connect(sock, (struct sockaddr *) &addr, sizeof(addr))
```

Abb. 21: Verbindung zum Server

Wenn etwas schiefgehen sollte wird ein entsprechender Fehler ausgegeben. Steht die Verbindung muss der Basis-Server sich zum beim IRC-Server registrieren (siehe Kapitel 2.3.3). Um sich zu registrieren müssen die benötigten Nachrichten in dem vom IRC Protokoll geforderten Format an den Server senden. Um eine Nachricht zu senden verwendet man die Funktion `send`.

```
send(sock, PROT, (int) strlen(PROT), 0);
send(sock, PASS, (int) strlen(PASS), 0);
send(sock, SRV, (int) strlen(SRV), 0);
```

Abb. 22: Senden der für die Registrierung benötigten Befehle

Nun ist der Basis-Server korrekt mit dem IRC-Server verbunden. Wir kopieren die lokale Variable `sock` in die globale Variable `mainsock`. Dieser Socket ist der Socket den von der ganzen Applikation verwendet wird um Nachrichten zu senden und zu empfangen. Nun können wir die einzelnen Dienste erstellen und ebenfalls verbinden:

5.4.2 Dienste

Da ein Dienst im Sinne des Protokoll ein Nutzer mit speziellen Rechten darstellt muss sich ein solcher lediglich als normaler Benutzer zum Server verbinden. Daher wird für jeden Dienst der NICK Befehl an den Server gesendet. Im UnrealIRCd Protokoll kann man statt NICK auch „&“ senden. Wir definieren `SNICK` als Konstante für den Befehl:

```
#define SNICK "& %s 1 0 %s %s %s 0 +qdS * :%s\r\n"
```

Abb. 23: Konstante für Dienst-Verbindung

und senden den Befehl mit den benötigten Parameter an den Server:

```
extern int ns_connect(int sock) {
    char *nick = (char*) malloc(sizeof(char) * 1024);
    sprintf(nick, SNICK, ns_name, s_user,
            s_host, s_name, ns_realname);
    int rc = send(sock, nick, (int) strlen(nick), 0);
    free(nick);
    return rc;
}
```

Abb. 24: Verbindung Dienst zu Basis-Server

Nun sind die Dienste entsprechend verbunden und können Befehle verarbeiten. Damit die Verbindung offen bleibt begibt sich die Applikation nun in eine Endlosschleife:

```
/* THIS IS THE MAIN LOOP */
while(!quitting)
{
    s = recv(mainsock,buf, sizeof(ircbuf),0);
    ...
}
```

Abb. 25: Endlosschleife um die Verbindung aufrecht zu erhalten

quitting ist eine Ganzzahlvariable die beim Serverstart auf 1 gesetzt wird. Solange dieser Wert auf bleibt.

Mit recv können Nachrichten über den Socket mainsocket empfangen werden⁹.

5.4.3 Verarbeitung von Befehlen durch den Basis-Server

Damit Befehle überhaupt an den jeweiligen Dienst weitergeleitet werden können müssen diese vom Basis-Server verarbeitet und weitergeleitet werden. Wie im vorherigen Kapitel beschrieben werden Befehle mit recv über den Socket empfangen. Wir wollen nun untersuchen wie diese Befehle über den Server empfangen werden und verarbeitet werden können.

Erinnern wir uns an das Kapitel 2.3.3.5. Wir haben dort festgehalten, dass ein Befehl an einen Dienst mit PRIVMSG gesendet werden kann:

```
PRIVMSG Empfänger :Nachricht CRLF
```

Da jeder Befehl in IRC mit Carriage Return / Line Feed (CRLF) abgeschlossen werden muss können wir so einen Befehl eindeutig als solchen identifizieren. Da über den Socket die Befehl unformatiert gesendet werden muss jeweils nur den Teil bis und mit CRLF betrachtet und verarbeitet werden. Mit der C-Funktion strtok kann eine Zeichenkette nach einem gewünschten Muster abgeschnitten werden:

```
s = recv(mainsock,buf, sizeof(ircbuf),0);
if(s)
{
    buf[s] = 0;
    char *pch = strtok(buf, "\r\n");
    while(pch!=NULL)
    {
        strcpy(ircbuf,pch);
        parse();
        ircbuf[s]=0;
        pch = strtok(NULL, "\r\n");
        addlog(1,ircbuf);
    }
}
```

Abb. 26: Aufteilung der empfangenen Zeichenkette in korrekt IRC Befehle

Mit **strtok** wird nun eine korrekte IRC-Zeichenkette in die Variable **pch** abgelegt und nach **ircbuf**, die eine globale Variable ist, abgelegt. Mit der Funktion **parse()** wird diese nun verarbeitet.

Die Befehle sollen möglichst dynamisch und nicht über if-then-else Vergleiche aufgerufen werden können. Um dies zu bewerkstelligen wird die Funktion **parse()** so ausgelegt, dass der zu verarbeitende IRC Befehl in drei Teile aufgeteilt wird:

- Der Name des Befehls der aufgerufen werden soll
- Die Anzahl Argumente
- Ein Datenfeld (Array) mit den Argumenten

```

void parse(void) {
    char source[1024], cmd[1024], buf[1024], *pch, **av;
    int ac;
    irc_cmd *ic;
    strscpy(buf, ircbuf, sizeof(buf));
    if (*buf == ':') {
        pch = strpbrk(buf, " ");
        if (!pch)
            return;
        *pch = 0;
        while (isspace(*++pch))
            ;
        strscpy(source, buf + 1, sizeof(source));
        strscpy(buf, pch, sizeof(buf));
    } else {
        *source = 0;
    }
    if (!*buf)
        return;
    pch = strpbrk(buf, " ");
    if (pch) {
        *pch = 0;
        while (isspace(*++pch))
            ;
    } else
        pch = buf + strlen(buf);
    strscpy(cmd, buf, sizeof(cmd));
    ac = tokenize(pch, &av);
    if ((ic = find_cmd(cmd))) {
        if (ic->func)
            ic->func(source, ac, av);
        if (strcmp(buf, "PING") != 0)
            addlog(1, LOG_DBG_SERVERMSG, s_unreal, ircbuf);
    } else
        addlog(2, APP_ERR_UNKNOWNMSG, ircbuf);
    free(av);
}

```

Abb. 27: Die Funktion parse()

Der Inhalt des Befehls wird nun anhand der Leerzeichen weiter aufgeteilt und die erste Zeichenkette als Befehl in der Variable **cmd** abgelegt. Um die Anzahl Argumente und die Argumente selbst ablegen zu können wird die Funktion **tokenize** verwendet. Diese Funktion teilt den Rest des Befehls weiter auf und legt die einzelnen Argumente in **av** ab.

Angemerkt sei hier, dass die Variable **av** noch nicht initialisiert wurde. Dies ist hier noch nicht benötigt. Wir übergeben die Speicheradresse von **av** mittels **&av** an die Funktion **tokenize**, die nach der Aufteilung die eigentliche Ablage der Werte übernimmt:

```

int tokenize(char *buf, char ***argv) {
    int argvsize = 8;
    int argc = 0;
    char *pch;
    *argv = smalloc(sizeof(char*) * argvsize);
    while (*buf)
    {
        if(argc == argvsize)
        {
            argvsize += 8;
            *argv = srealloc(*argv, sizeof(char*) * argvsize);
        }
        if (*buf == ':')
        {
            (*argv)[argc++] = buf+1;
            buf = "";
        }
        else
        {
            pch = strpbrk(buf, " ");
            if(pch)
            {
                *pch++ = 0;
                while(isspace(*pch))
                {
                    pch++;
                }
            }
            else
            {
                pch = buf + strlen(buf);
            }
            (*argv)[argc++] = buf;
            buf = pch;
        }
    }
    return argc;
}

```

Abb. 28: Die Funktion tokenize()

In der Funktion **tokenize** sehen wir, dass hier die Variable **av** als Zeiger auf das Datenfeld definiert wurde. Dies hat den Zweck, dass nicht eine neue lokale Variable **av** benutzt werden soll, sondern diejenige, die aus **parse()** genommen werden soll. Das funktioniert, da die Variable auf dem Stack abgelegt wird. Der Aufruf von **tokenize** wird ebenfalls auf den Stack gelegt (**push**) und nach der Abarbeitung von Stack entfernt (**pop**).

Wir befinden uns nun also wieder in der Funktion **parse()**. **Tokenize** hat uns die Anzahl Argumente zurückgeliefert und die Argumente selbst in die Variable **av** abgelegt.

Nun ist also alles vorhanden, was benötigt um den Befehl auszuführen. Um nun den Befehl mit den Argumenten direkt aufzurufen benötigen wir zunächst eine Struktur, die es uns erlaubt, den Befehl dynamisch aufzurufen.

```
struct _irc_command
{
    const char *name;
    void (*func)(char *source, int ac, char **av);
};
```

Abb. 29: Struktur dynamischer IRC-Befehl

So können wir anhand des Namens die dazugehörige Funktion mit den Argumenten aufrufen. Wir erstellen eine Variable die ein Mapping zwischen einer Zeichenkette und einem Befehl beinhaltet.

```
irc_cmd irc_cmds[] = {
    { "EOS", NULL },
    { "ERROR", NULL },
    { "JOIN", c_join },
    { "KICK", c_kick },
    { "KILL", c_kill },
    { "MODE", c_mode },
    { "NETINFO", NULL },
    { "NICK", c_nick },
    { "PASS", NULL },
    { "PART", c_part },
    { "PRIVMSG", c_privmsg },
    { "PROTOCTL", NULL },
    { "PING", c_ping },
    { "QUIT", c_quit },
    { "SERVER", NULL },
    { "SMO", NULL },
    { "TOPIC", c_topic },
};

irc_cmd *find_cmd(const char *name)
{
    irc_cmd *cmd;
    for (cmd = irc_cmds; cmd->name; cmd++)
    {
        if(stricmp(name, cmd->name) == 0)
            return cmd;
    }
    return NULL;
}
```

Abb. 30: IRC-Befehle

In der Variable `irc_cmds[]` ist diese Mapping. Wir rufen also die Funktion `find_cmd` auf und geben den Namen mit. Die Variable `irc_cmds` wird durchsucht und bei einem Treffer wird der dazugehörige Befehl aufgerufen.

Wir fassen also zusammen: Wir z.B. ein `PRIVMSG` Befehl an den Server gesendet erkennt die Applikation diesen Befehl und ruft den entsprechenden Befehl im Code auf.

Wir sehen also, dass im Falle von `PRIVMSG` also die Funktion `c_privmsg` aufgerufen werden soll.

5.4.4 Benutzer- und Channelverwaltung

Damit Berechtigungen und Stati verwaltet werden können muss der Server die Benutzer und die Channels verwalten. Zu diesem Zweck werden für jeden Benutzer der sich zum Server verbindet und für jeden Channel der geöffnet wird eine entsprechende Datenstruktur angelegt, der die Attribute speichert. Diese dienen nur der Verarbeitung und werden nicht in der Datenbank gespeichert.

5.5 Implementierung Dienste

5.5.1 Allgemein

Nachdem nun der Server läuft und die Dienste verbunden sind müssen nun die Dienste implementiert werden.

Da die Dienste grundsätzlich alle eine ähnliche Implementierung aufweisen und sich nur im Inhalt unterscheiden werden wir nun die grundsätzliche Funktionsweise beschreiben.

Wie im Basis-Server sollen auch für die Dienste die einzelnen Befehle dynamisch aufgerufen werden können. Die Dazu benötigten Datenstrukturen sind identisch mit der für den Basis-Server:

```
struct _ns_cmd
{
    const char *name;
    void (*func)(char *src,int ac,char **av);
};
```

Abb. 31: Datenstruktur für Dienst-Befehle

Dazu wird für jeden Dienst eine entsprechen Such-Funktion implementiert und eine Mapping-Variable angelegt die den Befehlsnamen dem Befehl zuweist.

Jeder Befehl der an einen Dienst gesendet wird muss zunächst validiert werden. Da in der Variable ac immer die Anzahl Argumente die übergeben wird gespeichert wird zunächst die korrekte Anzahl an Argumenten überprüft. Ist diese nicht korrekt wird eine Fehlermeldung ausgegeben mit dem Hinweis auf die entsprechende Hilfsfunktion:

```
if(ac<3) {
    notice(ns_name, src, NS_RPL_REG_USAGE);
    notice(ns_name, src, NS_RPL_HLP_SHORT, ns_name, "REGISTER");
    return;
}
```

Abb. 32: Falsche Anzahl Argumente

Danach werden immer die Argumente selbst validiert. Diese unterscheiden sich nach dem Inhalt, werden aber immer gleich behandelt. Ein Argument kann eine Unterfunktion oder ein Wert sein. Kann der Dienst mit dem Argument nichts anfangen wird ebenfalls eine entsprechende Fehlermeldung ausgegeben.

```
/* check for the correct parameters */
if ((!email || !strcmp(email, "")) || (!pass || !strcmp(pass, ""))) {
    notice(ns_name, src, NS_RPL_REG_USAGE);
    notice(ns_name, src, NS_RPL_HLP_SHORT, ns_name, "REGISTER");
    return;
}
```

Abb. 33: Ungültige Argumente

5.5.2 Nickserv

Ein Nickname soll eine Entität darstellen. Jeder Nickname hat diverse Attribute, daher bietet es sich an, diese in einer Datenstruktur darzustellen:

```
struct _nickinfo {  
    NickInfo *next, *prev;  
    int id;  
    int auth_chan;  
    int auth_notify;  
    unsigned int authcount;  
    myacc *accesslist;  
    unsigned short channelcount;  
    char *email;  
    int enforced;  
    int hidemail;  
    char *last_realname;  
    time_t last_seen;  
    const char *last_usermask;  
    unsigned short memomax;  
    int mforward;  
    char *mforward_to;  
    char *mlock;  
    short mnotify;  
    char nick[NICKMAX];  
    int noop;  
    notify *notifylist;  
    int nomemo;  
    char pass[PASSMAX];  
    int protect;  
    long reserved[4];  
    time_t time_reg;  
    char *url;  
    auth *authlist;  
};
```

Abb. 34: Struktur für Nickname

Das bedeutet, dass sämtliche Information eines Nickname in dieser Datenstruktur gekapselt werden.

5.5.2.1 Registrierung Nickname

Ein Nickname wird mit dem Befehl REGISTER registriert. Nickserv ruft dazu die Funktion `ns_register` auf. Dabei werden die Eingaben validiert. Sind alle Eingaben korrekt wird eine neue Struktur angelegt. Diese Struktur wird dann in eine Liste von Strukturen (verkettete Liste) abgelegt.

Diese globale Liste wird benötigt, damit ein bestimmter gesucht werden kann und die Nicknames in der Datenbank gespeichert werden können.

Sobald der Nickname registriert wurde wird automatisch der Benutzermodus `+r` (Registrierter Nickname) gesetzt und der Benutzer erhält ein Flag, dass dieser korrekt für den Nickname identifiziert ist.


```

NickInfo *register_nick(const char *src, const char *password, char *email) {
    user *u = finduser(src);
    NickInfo *n;
    char *usermask = (char*) malloc(sizeof(char*) * 1024);
    sprintf(usermask, "%s@%s", u->username, u->hostname);
    n = calloc(sizeof(NickInfo), 1);
    if (!src)
        src = "";
    strcpy(n->nick, src, NICKMAX);
    strcpy(n->pass, password, PASSMAX);
    n->email = strdup(email);
    n->nomemo = ns_no_memo;
    n->auth_chan = ns_auth_channel;
    n->auth_notify = ns_auth_notify;
    n->protect = ns_high_protect;
    n->hidemail = ns_hide_email;
    n->noop = ns_noop;
    n->last_realname = strdup(u->realname);
    n->last_seen = time(NULL);
    n->time_reg = time(NULL);
    n->last_usermask = strdup(usermask);
    n->mforward = 0;
    if (ns_autoaccess) {
        ns_access_add_mask(n, usermask);
    }
    n->next = nicklist;
    if (nicklist)
        nicklist->prev = n;
    nicklist = n;
    return n;
}

```

Abb. 35: Ablage der Attribute in einer Struktur

5.5.2.2 Löschen eine Nickname

Soll ein Nickname gelöscht werden, wird dieser einfach aus der verketteten Liste entfernt:

```

void delete_nick(NickInfo *n) {
    if (n->prev)
        n->prev->next = n->next;
    else
        nicklist = n->next;
    if (n->next)
        n->next->prev = n->prev;
    free(n);
}

```

Abb. 36: Löschen eines Nicks

5.5.2.3 Identifikation

Wenn sich nun ein Benutzer zum Server verbindet und einen Nickname benutzt der registriert ist wird er zur Identifikation aufgefordert. Es stehen drei Schutz-Stufen zur Verfügung. Wird der Schutz auf OFF gesetzt, darf der Benutzer den Nickname benutzen, jedoch kann er natürlich keine Änderungen daran vornehmen.

Ist der Schutz auf normal gestellt, hat der Benutzer 60 Sekunden Zeit sich per Passwort zu identifizieren. Tut er das nicht, so wird der Nickname gesperrt und der Benutzer erhält einen „Guest“ Nickname mit einer Zufallsnummer

5.5.2.4 Timer

Um einen solchen Timer zu implementieren, greifen wir auf Signale zurück. Wir starten beim Programmstart den Timer indem wir ein neues Signal anlegen:

```
char buf[10000];
if(signal(SIGALRM, timer_event_handler)==SIG_ERR)
{
    printf("Error message: %s\n", strerror(errno));
    addlog(2, "Error in signal()\n");
    return;
}
```

Abb. 37: Signal

Wir entscheiden uns für den Signaltyp SIGALRM, weil mit diesem Typ das Signal beim Ablauf eines definierten Timers gesendet wird und es abgefangen werden kann¹⁰.

Wir schreiben nun eine Funktion `set_timer`, der festlegt, in welchen Abständen das Signal SIGALRM gesendet werden soll:

```
void set_timer(time_t period_in_secs) {
    struct itimerval timer_val;
    bzero(&timer_val, sizeof(timer_val));
    timer_val.it_value.tv_sec = period_in_secs;
    timer_val.it_interval.tv_sec = period_in_secs;
    if (setitimer(ITIMER_REAL, &timer_val, NULL) != 0)
        perror("Error in setitimer()");
}
```

Abb. 38: set_timer

Beim Anlegen des Signals haben wir die Handler-Funktion `timer_event_handler` angegeben. Diese wird nun in dem Abstand, den wir in der Funktion `set_timer` festgelegt haben, aufgerufen.

```
void timer_event_handler(int sigid) {
    if (sigid == SIGALRM) {
        check_timeouts();
        check_connections();
        check_expiry();
        check_akills();
        check_save();
    }
}
```

Abb. 39: timer_event_handler

In der Funktion `check_timeout` werden die Identifikationstimer, die als Datenstruktur angelegt wurden, überprüft.

5.5.2.5 Weitere Nickserv Funktionen

Für Nickserv wurden die gewünschten Funktionen implementiert. Da die detaillierte Beschreibung der Funktionen den Rahmen dieser Arbeit sprengen würde, verzichten wir an dieser Stelle darauf und verweisen auf den Quellcode.

Allgemein halten wir fest, dass für sämtliche Funktionen die benötigten Datenstrukturen als verkettete Listen angelegt und die Verarbeitung immer nach dem gleichen Muster erfolgt.

5.5.3 Chanserv

Genau wie Nickserv sollen die registrierten Channels in einer Datenstruktur festgehalten werden. Bei der Registrierung wird eine solche Struktur angelegt und in eine entsprechende verkettete Liste eingefügt.

Für jeden Befehl der über Chanserv abgesetzt wird, muss zunächst überprüft werden, ob der Benutzer überhaupt berechtigt ist, diesen Befehl zu verwenden.

5.5.3.1 Operatoren-Listen

Wie in den Anforderungen beschrieben soll Chanserv diverse Arten von Operatoren-Listen unterstützen. Um Wiederholungen im Code zu vermeiden (Programmierungs-Prinzip Don't repeat yourself, DRY) stellen wir eine Funktion zur Verfügung, die für alle Arten von Operatoren gelten soll.

```
int cs_xop_get_level(user *u, ChanInfo *c);
void cs_xop_add(char *src, char *chan, int list, char *nick);
void cs_xop_del(char *src, char *chan, int list, char *nick);
void cs_xop_list(char *src, char *chan, int list);
void cs_xop_wipe(char *src, char *chan, int list);
```

Abb. 40: Funktionsprototypen für Operatorenlisten

Die Berechtigung für den jeweiligen Befehl wird wie folgt überprüft:

```
int cs_xop_get_level(user *u, ChanInfo *c) {
    if(u->oper>cs_admin_access) {
        return ACCESS_SRA;
    }
    usernick *un = u->usernick;
    int level = 0;

    struct cschans *uc = u->cschans;
    while(uc) {
        if((strcmp(uc->channel, c->name) == 0) && (uc->level == CHAN_IDENTIFIED)) {
            return ACCESS_FND_FULL;
        }
        uc = uc->next;
    }
    while(un) {
        if(un->level == 2) {
            return get_access_for_nick(c, un->n);
        }
        un = un->next;
    }

    return level;
}
```

Abb. 41: Überprüfung der Berechtigung für einen Chanserv Befehl

Verfügt der Benutzer über keine Rechte, wird 0 zurückgegeben, ansonsten die Zahl, die der Berechtigung entspricht.

Sämtliche Operatoren-Berechtigungen werden in einer globalen Variablen abgelegt, um die Suche zu vereinfachen.

5.5.4 Opserv

Für Opserv ist keine komplizierte Logik erforderlich. Benutzer die noch keine Berechtigung für URC-Operatoren Befehle durch den Server haben können hinzugefügt werden und der Zugriff kann für jeden Befehl einzelnen gesetzt werden. Implementiert sind die Einzel-Berechtigung als 1 oder 0 in der entsprechenden Struktur.

5.5.4.1 AKILL

Der Autokill Befehl wird als Liste definiert. Wenn ein berechtigter Benutzer diesen Befehl ausführt wird ein Eintrag in diese Liste geschrieben. Da für diesen Befehl die Zeitdauer angegeben werden kann benötigen wir eine Implementierung, die es uns erlaubt, Kürzel wie 1w für eine Woche, oder 2y für zwei Jahre zu verwenden. Um dies zu bewerkstelligen bedienen wir uns regulären Ausdrücken:

```
#define TIME_FORMAT_H "[[:digit:]]+h"
#define TIME_FORMAT_M "[[:digit:]]+m"
#define TIME_FORMAT_W "[[:digit:]]+w"
#define TIME_FORMAT_D "[[:digit:]]+d"
#define TIME_FORMAT_Y "[[:digit:]]+y"
```

Abb. 42: Reguläre Ausdrücke für Zeitkürzel

Diese erkennen, ob eine gültige Zeitangabe definiert wurde. Mit folgendem Ausschnitt berechnen wir dann die entsprechende Zeit in Minuten:

```
}
if(match(dur, TIME_FORMAT_H)) {
    return (atol(dur) * MINUTES_PER_HOUR);
} else if(match(dur, TIME_FORMAT_D)) {
    return (atol(dur) * MINUTES_PER_DAY);
} else if(match(dur, TIME_FORMAT_W)) {
    return (atol(dur) * MINUTES_PER_WEEK);
} else if(match(dur, TIME_FORMAT_M)) {
    return (atol(dur) * MINUTES_PER_MONTH);
} else if(match(dur, TIME_FORMAT_Y)) {
    return (atol(dur) * MINUTES_PER_YEAR);
}
return 0;
```

Abb. 43: Berechnung Zeitangabe

5.5.4.2 Weitere Opserv-Befehle

Die weiteren Opserv-Befehle leiten lediglich den entsprechenden Befehl an den Server weiter der diesen dann verarbeitet.

5.5.5 Botserv

Ein Bot kann mit `bs_add` erfasst werden. Datentechnisch ist ein Bot eine Struktur mit Attributen. Die Implementierung dieses Dienstes ist relativ simpel. Sobald der Bot hinzugefügt wurde wird eine Struktur angelegt und eine Liste eingefügt. Die einzelnen Befehle sind lediglich Weiterleitungen an den IRC-Server, der die entsprechenden Befehle ausführt.

5.5.6 Adminserv

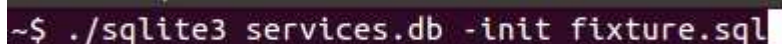
Adminserv besteht aktuell nur aus zwei Befehlen; `squit` und `savedata`. `Squit` bringt die Applikation zum stoppen und `savedata` löst eine Datenbankspeicherung aus. Details zur Datenspeicherung betrachten wir im nächsten Kapitel.

5.6 Datenbank

Wie bereits erwähnt benutzen die IRC Service eine SQLite Datenbank, da diese über eine API für C verfügt. Wir wollen auf den nächsten Seiten beschreiben wie wir mit der Datenbank umgehen wollen.

5.6.1 Erstellen Datenbank

Um alle benötigten Tabellen zu erstellen laden wir manuelle eine SQL Datei. Dazu benutzen wir das `sqlite` Tool, das wir im Projekt mitliefern.

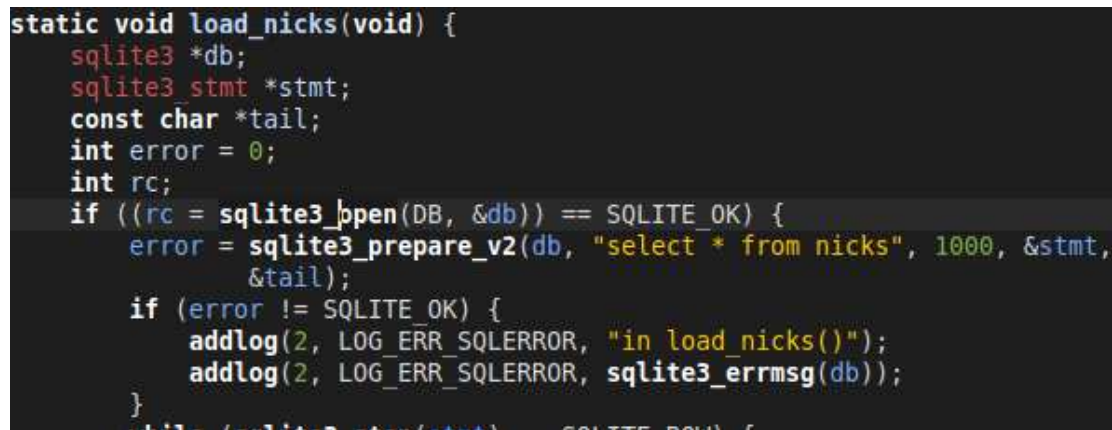


```
~$ ./sqlite3 services.db -init fixture.sql
```

Abb. 44: Erstellen der Tabellen

5.6.2 Laden der gespeicherten Daten

Beim Serverstart sollen die Daten von der Datenbank in den Arbeitsspeicher geladen werden. Dazu laden wir nacheinander die Tabellen der einzelnen Dienste. Beispielhaft wollen anhand der Nickserv Tabellen zeigen wir das funktioniert. Wir erstellen zunächst eine Datenbankverbindung:



```
static void load_nicks(void) {
    sqlite3 *db;
    sqlite3_stmt *stmt;
    const char *tail;
    int error = 0;
    int rc;
    if ((rc = sqlite3_open(DB, &db)) == SQLITE_OK) {
        error = sqlite3_prepare_v2(db, "select * from nicks", 1000, &stmt,
                                   &tail);
        if (error != SQLITE_OK) {
            addlog(2, LOG_ERR_SQLERROR, "in load_nicks()");
            addlog(2, LOG_ERR_SQLERROR, sqlite3_errmsg(db));
        }
    }
    while (sqlite3_step(stmt) == SQLITE_ROW) {
```

Abb. 45: Vorbereiten SQLite Datenbankverbindung

Wir sehen hier, dass auch die benötigte SQL Query schon angegeben wird. Mit `sqlite3_prepare_v2` bereiten wir das benötigte Statement vor und können nun durch die einzelnen Resultate durchgehen:

```

while (sqlite3_step(stmt) == SQLITE_ROW) {
    NickInfo *n = calloc(sizeof(NickInfo), 1);
    n->id = sqlite3_column_int(stmt, 0);
    strcpy(n->nick, (char*) sqlite3_column_text(stmt, 1), NICKMAX);
    strcpy(n->pass, (char*) sqlite3_column_text(stmt, 2), PASSMAX);
    n->last usermask = strdup((char*) sqlite3_column_text(stmt, 3));
}

```

Abb. 46: Resultate

Wir legen also für jede Zeile, die von der Tabelle zurückgeliefert wird eine neue Struktur für einen Nickname und weisen die Attribute entsprechend zu.

Nachdem alle Daten geladen wurden müssen wir die Verbindung wieder schliessen:

```

}
sqlite3_close(db);

```

Abb. 47: Schliessen der Datenbankverbindung

5.6.3 Speichern der Daten

Die Daten müssen in regelmässigen Abständen gespeichert werden. Da Änderungen der Daten nur im Arbeitsspeicher und nicht direkt in der Datenbank vorgenommen werden muss immer der gesamte Datenbestand gespeichert werden. Da immer etwas schiefgehen kann verwenden wir Transaktionen, das heisst es wird entweder alles oder nicht in die Datenbank geschrieben.

```

void db_save_nicks(void) {
    sqlite3 *db;
    int query_result = 0;
    int rc;
    if ((rc = sqlite3_open(DB, &db)) != SQLITE_OK) {
        addlog(2, LOG_ERR_SQLERROR, sqlite3_errmsg(db));
        return;
    }
    sqlite3_exec(db, "BEGIN", 0, 0, 0);
    sqlite3_exec(db, "DROP TABLE IF EXISTS NICKS", 0, 0, 0);
    sqlite3_exec(db, ns_create_nicks_table, 0, 0, 0);
    NickInfo *n = nicklist;
    while (n) {
        if (!(query_result = db_add_nick(db, n))) {
            addlog(2, "Error in db_add_nick, rolling back");
            sqlite3_exec(db, "ROLLBACK", 0, 0, 0);
            sqlite3_close(db);
            return;
        }
        n = n->next;
    }
    sqlite3_exec(db, "COMMIT", 0, 0, 0);
    sqlite3_close(db);
    return;
}

```

Abb. 48: Datenbank-Transaktion

Genau wie beim Laden erstellen wir zunächst eine Datenbank-Verbindung zur Datenbankdatei. Dann starten wir die Transaktion mit BEGIN.

Nun werden alle Speicher-Queries ausgeführt. Wenn nur eine Query fehlschlägt wird mit ROLLBACK die ganze Transaktion abgebrochen und der ursprüngliche Zustand wiederhergestellt. Ansonsten werden mit COMMIT die Resultate in die Datenbank geschrieben.

5.7 Hilfe

Jeder Dienst muss eine Hilfefunktion besitzen. Mit dem HELP Befehl soll zu jedem Befehl und unterbefehl ein Hilfetext angezeigt werden können.

Das Konzept ist schnell erklärt: Wird ein Hilfetext gefunden wird dieser aus der entsprechenden Hilfe-datei gelesen und als NOTICE ausgegeben, ansonsten wird eine Fehlermeldung angezeigt. Die Hilfe-dateien sind reine Text-Dateien die von der Applikation Zeile für Zeile gelesen und ausgegeben werden.

6 Testing

6.1 Konzept

Die Installation eines Unit Test Frameworks erwies sich als äusserst mühsam und daher wurde darauf verzichtet. Stattdessen sollte jede Funktion auf jede mögliche Eingabe getestet werden.

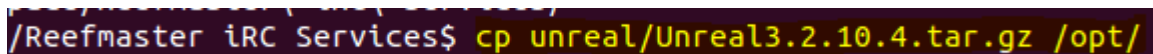
6.2 Test-Protokoll

Es wurde kein ausführliches Test-Protokoll erstellt. Jedoch wurden alle Funktionen auf alle möglichen Eingaben getestet. Bei der Entwicklung in der Programmiersprache C kommt es zudem immer wieder zu Segmentierungsfehlern. Um das Debugging zu erleichtern wurde das Tool valgrind benutzt. Mit valgrind kann die Applikation normal gestartet werden. Tritt jedoch ein Fehler auf stellt valgrind ausführliche Analyse-Resultate zur Verfügung die den Entwickler bei der Fehlersuche unterstützen.

6.3. Testing der Applikation

Um die Applikation testen zu können muss zunächst der UnrealIRCd Server installiert werden. Ein Archiv liegt diesem Projekt bei und wir wollen die Installation erläutern:

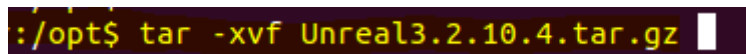
Kopieren des Archivs an einen geeigneten Ort, z.B. /opt/:



```
/Reefmaster iRC Services$ cp unreal/Unreal3.2.10.4.tar.gz /opt/
```

Abb. 49: Kopieren des Archivs

Als nächstes muss das Archiv entpackt werden:



```
:/opt$ tar -xvf Unreal3.2.10.4.tar.gz
```

Abb. 50: Entpacken des Archivs

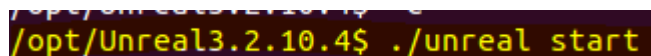
Danach wechseln wir ins das neu angelegte Verzeichnis und rufen den Befehl **Config** auf:



```
/Unreal3.2.10.4$ ./Config
```

Abb. 51: Konfiguration

Bei der Konfiguration kann alles einfach mit Enter bestätigt werden. Danach muss noch **make** ausgeführt werden um die Installation fertig zu stellen. Da UnrealIRCD konfiguriert werden muss, steht eine gültige Beispieldatei im Order „unreal“ bereit. Diese muss in das Hauptverzeichnis des Unreal-Servers kopiert werden. Danach kann der Unreal-Server wie folgt gestartet werden:



```
/opt/Unreal3.2.10.4$ ./unreal start
```

Abb. 52: Starten des UnrealIRCd

Nun läuft der Server und wir können uns den Services widmen. Im Hauptverzeichnis der Services muss nun „make“ ausgeführt werden, um die Applikation zu erstellen. Danach können wir mit ./services start die Service starten.

```
fish-guts@reefmaster:~/workspace/Reefmaster iRC Services$ ./services start
#####
#                                     #
#  Reefmaster                       #
#  IRC Services v 1.0               #
#                                     #
#####
*** Loading Configuration: services.conf...
```

Abb. 53: Starten der Services

Als IRC-Client wird der X-Chat in Linux empfohlen. Auf Ubuntu kann dieser wie folgt installiert werden:

```
$ sudo apt-get install xchat
```

Abb. 54: Installation Chat-Client

Nun kann man mit dem IRC-Client auf localhost verbinden und anfangen zu testen.

Um Befehle als Administrator zu testen wurde im UnrealIRCd ein Operatoren Benutzer vorbereitet, Benutzername und Passwort ist Admin.

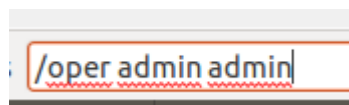


Abb. 55: Administratoren-Status erhalten

Um entsprechende Tests als IRC Operator durchzuführen wurde ein ebenfalls ein entsprechender Benutzer bereitgestellt. Benutzername und Passwort ist **ircop**

Auf weitere Hilfe wird an dieser Stelle verzichtet, da auch die Nützlichkeit der Hilfsfunktionen erprobt werden sollen.

7 Ausblick

Die Services sind schon sehr ausgebaut und können sehr viel. Dennoch sind in der Zukunft einige Modifikationen denkbar. So wird derzeit nur Englisch unterstützt. Denkbar ist eine Unterstützung für weitere Sprachen.

Adminserv unterstützt heute nur zwei Befehle. Je nach Resonanz der Benutzer kann Adminserv noch ausgebaut werden.

Die wichtigste Neuerung in zukünftigen Versionen ist bereits angedacht und in Planung: Ein Dienst zum Versenden von Kurznachrichten, Memoserv. Derzeit laufen Abklärungen, was Memoserv alles mitbringen sollte.

Da Nickserv und Chanserv bereits für diesen Dienst vorbereitet wurden dürfte sich die Komplexität der Umsetzung in Grenzen halten.

So oder so: Software lebt und darf niemals stillstehen. Anforderungen verändern sich laufend und Software muss den veränderten Anforderungen angepasst werden.

Damit entsprechendes Feedback von den Benutzern kommt soll das Projekt auf Sourceforge, einer Plattform für Open Source Software, zum Download bereitgestellt werden.

8 Fazit

Dieses Projekt war für eine Semesterarbeit sicherlich etwas gross angelegt. Da aber beim Autor dieser Arbeit ein ausgesprochenes Interesse am Thema vorhanden fiel es dennoch leicht das Projekt umzusetzen.

Technische Schwierigkeiten waren kaum vorhanden, da der Autor vom Beginn weg genaue Vorstellungen davon hatte, wie das Projekt zu implementieren sei.

Als etwas langwierig erwies sich die Erfassung der Anforderungen. Dennoch ist dies ein unverzichtbar Teil eines jeden Software-Projekt daher wurde speziell auch darauf Augenmerk gelegt.

Abschliessen möchte der Autor sich für das Interesse und den Input bedanken.

9 Anhang

9.1 Anhang A: Bilderverzeichnis

Abb. 1: Chatraum auf irc.freenode.org	9
Abb. 2: Schema eines IRC Netzwerkes	12
Abb. 3: Struktur eines IRC Netzwerkes.....	14
Abb. 4: Verbindungseinstellungen mIRC.....	16
Abb. 5: Rückgabe einer WHOIS Nachricht.....	17
Abb. 6: Use Case Basis Server.....	22
Abb. 7: Use Case Nickserv	26
Abb. 8: Use Case Chanserv	34
Abb. 9: Use Case Operserv	45
Abb. 10: Use Case Botserv.....	49
Abb. 11: Use Case Adminserv.....	54
Abb. 12: Architektur	56
Abb. 13: Datenbank-Schema.....	57
Abb. 14: Konfiguration Unrealircd.....	58
Abb. 15: Beispiel-Konfiguration Link	58
Abb. 16: Beispiel Basis-Server Konfiguration	59
Abb. 17: config_load.....	59
Abb. 18: Validierung der Unterblöcke.....	60
Abb. 19: Anlegen eines Sockets.....	60
Abb. 20: Konfiguration Socket und Adresse.....	61
Abb. 21: Verbindung zum Server.....	61
Abb. 22: Senden der für die Registrierung benötigten Befehle	61
Abb. 23: Konstante für Dienst-Verbindung.....	61
Abb. 24: Verbindung Dienst zu Basis-Server	61
Abb. 25: Endlosschleife um die Verbindung aufrecht zu erhalten.....	62
Abb. 26: Aufteilung der empfangenen Zeichenkette in korrekt IRC Befehle.....	62
Abb. 27: Die Funktion parse().....	63
Abb. 28: Die Funktion tokenize()	64
Abb. 29: Struktur dynmischer IRC-Befehl.....	65
Abb. 30: IRC-Befehle.....	65
Abb. 31: Datenstruktur für Dienst-Befehle	66
Abb. 32: Falsche Anzahl Argumente.....	66
Abb. 33: Ungültige Argumente.....	66
Abb. 34: Struktur für Nickname.....	67
Abb. 35: Ablage der Attribute in einer Struktur	68
Abb. 36: Löschen eines Nicks	68
Abb. 37: Signal.....	69
Abb. 38: set_timer.....	69
Abb. 39: timer_event_handler	69
Abb. 40: Funktionsprototypen für Operatorenlisten	70
Abb. 41: Überprüfung der Berechtigung für einen Chanserv Befehl.....	70
Abb. 42: Reguläre Ausdrücke für Zeitkürzel.....	71
Abb. 43: Berechnung Zeitangabe	71
Abb. 44: Erstellen der Tabellen	72

Abb. 45: Vorbereiten SQLite Datenbankverbindung.....	72
Abb. 46: Resultate	73
Abb. 47: Schliessen der Datenbankverbindung	73
Abb. 48: Datenbank-Transaktion	73
Abb. 49: Kopieren des Archivs.....	75
Abb. 50: Entpacken des Archivs	75
Abb. 51: Konfiguration	75
Abb. 52: Starten des UnrealIRCd	75
Abb. 53: Starten der Services	76
Abb. 54: Installation Chat-Client	76
Abb. 55: Administratoren-Status erhalten	76

9.2 Anhang B: Tabellenverzeichnis

Tabelle 1: Stakeholder.....	22
Tabelle 2: R-S-001: Unterstützte IRC Server.....	23
Tabelle 4: R-S-002: Konfiguration Basis-Server	23
Tabelle 5: R-S-003: Starten des Servers.....	23
Tabelle 6: R-S-004: Validierung Server-Konfiguration.....	24
Tabelle 7: R-S-005: Daten von Datenbank laden.....	24
Tabelle 8: R-S-006: Verbindung zum IRC Server.....	24
Tabelle 9: R-S-007: Dienste starten und Verbinden.....	24
Tabelle 10: R-S-008: Daten in regelmässigen Abständen speichern	25
Tabelle 11: Stakeholder Nickserv	26
Tabelle 12: Akteure Nickserv.....	26
Tabelle 13: R-NS-001: Nickserv Hauptfunktion	27
Tabelle 14: R-NS-002: Konfiguration Nickserv	28
Tabelle 15: R-NS-003: ACC.....	29
Tabelle 16: R-NS-004: ACCESS	29
Tabelle 17: R-NS-005: AUTH	30
Tabelle 18: R-NS-006: DROP	30
Tabelle 19: R-NS-007: GETPASS.....	30
Tabelle 20: R-NS-008: GHOST	31
Tabelle 21: R-NS-009: IDENTIFY	31
Tabelle 22: R-NS-010: INFO	31
Tabelle 23: R-NS-011: LIST.....	31
Tabelle 24: R-NS-012: LISTCHANS	32
Tabelle 25: R-NS-013: NOTIFY	32
Tabelle 26: R-NS-014: REGISTER.....	32
Tabelle 27: R-NS-015: RELEASE	33
Tabelle 28: R-NS-016: SET	33
Tabelle 29: R-NS-016: SETPASS	33
Tabelle 30: R-NS-017: Identifikationstimer	33
Tabelle 31: R-CS-002: Konfiguration Chanserv.....	37
Tabelle 32: R-CS-002: ACC.....	37
Tabelle 33: R-CS-003: AKICK.....	38
Tabelle 34: R-CS-004: AOP.....	38
Tabelle 35: R-CS-005: DE-/HALFOP	38
Tabelle 36: R-CS-006: DE-/OP.....	39

Tabelle 37: R-CS-007: DEVOICE	39
Tabelle 38: R-CS-008: DROP	39
Tabelle 39: R-CS-009: GETPASS	40
Tabelle 40: R-CS-010: HOP	40
Tabelle 41: R-CS-011: IDENTIFY	40
Tabelle 42: R-CS-013: INVITE	41
Tabelle 43: R-CS-013: LIST	41
Tabelle 44: R-CS-014: MDEOP	41
Tabelle 45: R-CS-015: MKICK	41
Tabelle 46: R-CS-016: REGISTER	42
Tabelle 47: R-CS-017: SET	42
Tabelle 48: R-CS-018: SETPASS	43
Tabelle 49: R-CS-019: SOP	43
Tabelle 50: R-CS-020: UNBAN	43
Tabelle 51: R-CS-021: UOP	44
Tabelle 52: R-CS-022: Uop	44
Tabelle 53: R-OS-001: Konfiguration Opserv	46
Tabelle 54: R-OS-002: AKILL	46
Tabelle 55: R-OS-003: CHATOPS	47
Tabelle 56: R-OS-004: CHGHOST	47
Tabelle 57: R-OS-005: GLOBAL	47
Tabelle 58: R-OS-006: KILL	48
Tabelle 59: R-OS-007: LOCAL	48
Tabelle 60: R-OS-008: OPER	48
Tabelle 61: R-OS-009: SGLINE / SKLINE / SZLINE / SQLINE	49
Tabelle 62: R-OS-001: Konfiguration Botserv	51
Tabelle 63: R-BS-002: ADD/DEL	51
Tabelle 64: R-BS-003: DE-/HALFOP	51
Tabelle 65: R-BS-004: DE-/VOICE	52
Tabelle 66: R-BS-005: DE-/OP	52
Tabelle 67: R-BS-006: GETPASS	52
Tabelle 68: R-BS-007: IDENTIFY	52
Tabelle 69: R-BS-008: INFO	53
Tabelle 70: R-BS-009: KICK	53
Tabelle 71: R-BS-010: LIST	53
Tabelle 72: R-BS-011: MSG	53
Tabelle 73: R-BS-012: SET	54
Tabelle 74: R-BS-013: SETPASS	54
Tabelle 75: R-AS-001: Konfiguration Adminserv	55
Tabelle 76: R-AS-002: SAVEDATA	55

9.4. Literaturverzeichnis

- [1] http://www.selflinux.org/selflinux/html/irc_geschichte01.html abgerufen 01.01.2015
- [2] https://de.wikipedia.org/wiki/Internet_Relay_Chat#Entwicklung, abgerufen 01.01.2015
- [3] <http://www.at-mix.de/internet/internet-0207.htm>, abgerufen am 01.01.2015
- [4] <http://www.webmaster.com/crtabeditions.htm>, abgerufen am 02.01.2015
- [5] <https://www.unrealircd.org/files/docs/unreal32docs.html#ulinesblock>, abgerufen am 21.02.2014

- [6] Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language. Englewood Cliffs, Vereinigte Staaten: Prentice Hall, 50. Auflage, 2012, Seite 177. ISBN 0-13-119362-8
- [7] Kernighan, Brian W., Ritchie, Dennis M.: The C Programming Language. Englewood Cliffs, Vereinigte Staaten: Prentice Hall, 50. Auflage, 2012, Seite 14. ISBN 0-13-119362-8
- [8] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 590. ISBN 978-0-321-63773-4
- [9] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 612. ISBN 978-0-321-63773-4
- [10] Stevens, Richard W., Rago, Stephen A.: Advanced Programming in the UNIX Environment. Upper Saddle River, Vereinigte Staaten: Addison-Wesley, 1. Auflage, 2013, Seite 317. ISBN 978-0-321-63773-4
- Titelbild: <http://www.sankt-mauritz.com/sites/default/files/bilder/nachricht/netzwerk.jpg>, abgerufen am 29.10.2014s