Anforderungen an das Kommunikationssystem

Hier wird die Schnittstelle zwischen der Applikation und dem Kommunikationssubsystem sowie das Verhalten des Kommunikationssystems an dieser Schnittstelle festgelegt. Diese Anforderungen sollen der Problemstellung genügen und bilden die Grundlage für die Entwicklung des Systems

Architektur: ---

A1: Pro Benutzer darf es höchstens eine Anwendung geben.

A2: Pro Knoten können mehrere Benutzer existieren.

A3: Pro Benutzer darf es mehrere private Chat-Kanäle geben.

EinLoggen --

Bemerkung: Alle SP ServicePrimitive werden mit .req bzw. .ind abgeschlossen.

L1: Eine Anwendung meldet sich mit dem Nahmen des Benutzers (nickName) bei dem System über die Dienstprimitive *SP_login.req(nickName: nickNameType, password: passwordType)* und erhält eine Bestätigung *SP_login_acc.ind* (*SP_login_acc.ind*) beim Erfolg bzw. *SP_login_ref.ind(reason:reasonType)* anderenfalls.

Achtung: im weiteren werden die .req und .ind weggelassen. Mit ***.req*** enden alle Signale, die von der Umgebung (Application) zu dem System geschickt werden. Mit ***.ind*** enden alle Signale, die zu Umgebung (Applicatoin) geschickt werden.

L2: Die Anwendung meldet sich ab mit der Dienstprimitive *SP_logout*.

L3: Ein Benutzer kann sich nur einmal beim System anmelden. Stellt das Kommunikationssystem fest, daß der Benutzer bereits angemeldet ist, wird dies durch die Dienstprimitive *login_ref(reason: reasonType) (reason = "alreadyLoggedIn")* bekanntgegeben. Mit *SP_relogin(nickName: nickNameType, password: passwordType)* kann der Benutzer sich erneut anmelden. Alle bis dahin laufende Kommunikaitonsbeziehungen alter login-session z.B. Privat-Kanäle werden dabei terminiert. Die eventuell noch existierende alte Anwendung wird durch die Dienstprimitive SP_*ConnectionBroken(reason: reasonType)* (reason = "relogin") informiert und muß terminieren.

L4: Die Anwendung kann mit *SP_who* eine Liste der zu dieser Zeit angemeldeten Personen anfragen und bekommt diese mit *SP_who_resp(wholist: whoListType)*.

OEffentlicher-Kanal ---------------------------------------

OEK1: Die Anwendung kann einen öffentlichen Kanal zu allen angemeldeten Benutzern öffnen. Dies wird durch die Dienstprimitive *SP_open_chat_all* gemacht.

OEK2: Die Anwendung kann diesen öffentlichen Chat-Kanal mit der Dienstprimitive *SP_close_chat_all* schließen. Es muß ein weicher Verbindungsabbau sein. (Alle zuvor gesendeten Daten müssen übertragen werden)

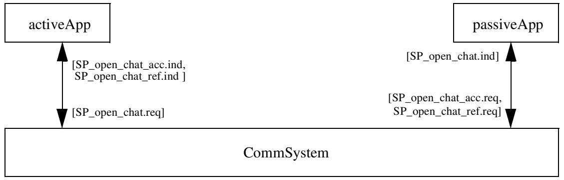
OEK3: Bei einem etablierten öffentlichen Chat-Kanal ist es möglich, Textdaten mit der Dienstprimitive *SP_say(textData: textDataType, selfName: nickNameType)* an alle zu diesem Zeitpunkt angemeldeten Benutzer zu übertragen.

OEK4:

Bei einem etablierten öffentlichen Chat-Kanal wird dieAnwenung mit dem Signal SP_say*(textData: textDataType, nickName: nickNameType)* benachrichtigt.

PRivater-Kanal ---

PRK1: Die Anwendung kann einen privaten bidirektionalen Chat-Kanal zu einem der angemeldeten Benutzer aufbauen. Dies wird durch die Dienstprimitive *SP_open_chat (nickName: nickNameType)* angestoßen, mit *SP_open_chat(nickName:nickNameType)* der Applikation auf der Gegenseite mitgeteilt, und mit der Dienstprimitive *SP_open_chat_acc* bzw. *SP_open_chat_ref(reason: reasonType)* beantwortet. Zur Verdeutlichung hier ein Diagramm



PRK2: Jede der beiden Anwendungen kann diesen privaten Chat-Kanal mit der Dienstprimitive *SP_close_chat(nickName:nickNameType)* wieder schließen. Aus der Sicht des Senders muß es eine weiche Verbindungsabbau sein. (Alle zuvor gesendeten Daten müssen übertragen werden). Der Kommunikationspartner wird mit *SP_close_chat(nickName:nickNameType)* benachrichtigt. Das Parameter *nickName:nickNameType* enthält immer der *NickName* des jeweiligen Kommunikationspartners.

PRK3: Bei einem etablierten privaten Chat-Kanal ist es möglich, Textdaten mit der Dienstprimitive *SP_tell(nickName:nickNameType, selfName:nickNameType, textData: textDataType)* an den entsprechenden Benutzer zu übertragen.
