PRIAMOS

Dokumentation des Message-Systems

Guido Appenzeller
July 3, 1995

Institut für Prozessrechentechnik und Robotik
Prof. Dr.-Ing. U. Rembold
Prof. Dr.-Ing. R. Dillmann
Universität Karlsruhe
Fakultät für Informatik

Contents

1 Das RobMsg Messaging-System

Die verschiedenen Komponenten des Robotersystems PRIAMOS verwenden zur Kommunikation untereinander sehr unterschiedliche Standards. Durch das hier realisierte Message-System, genannt RobMsg-System, soll eine einheitliche Schnittstelle in den verschiedenen Modulen zur Verfügung gestellt werden mit dem Nachrichtenohne Rücksichtnahme auf die Beschaffenheit der Verbindung übertragen werden können.

Desweiteren wird auf die hierzu notwendige Verbesserungen und Änderungen der Funkstrecke eingegangen.

Die Teile von Priamos, die an der Kommunikation teilnehmen, lassen sich in vier Gruppen einteilen:

- Die unter PVM auf der Workstation laufenden Teile von MARS.
- Die auf der Fahrzeug-E7 laufenden Prozesse.
- Die auf der VISION-E7 laufenden Prozesse.
- Das auf einer Workstation laufende Programm STEREO.

Diese vier Gruppen benutzen zur Kommunikation untereinander grundsätzlich verschiedene Nachrichtenträger:

- PVM-Pipes zwischen den einzelnen PVM-Prozessen
- Sockets über TCP/IP zwischen STEREO und der VISION-E7 sowie zwischen den beiden E7-Karten.
- Eine Funkmodemstrecke zwischen `radio_comm` unter MARS und der Fahrzeug-E7

Die Socketverbindung und die PVM-Pipes haben eine sehr gute Datensicherheit. Nach der Programmierung eines sicheren Protokolls auch auf der Funkstrecke konnte das RobMsg-System ohne jede eigene Fehlererkennung implementiert werden.

Der Benutzer (d.h. der Programmierer der das RobMsg-System benutzt) gibt lediglich den Empfänger einer Nachricht an und übergibt die Nachricht einer Routine. Die Übermittlung der Daten, eventuell auch über mehrere Stationen übernimmt vollständig das RobMsg-System.

Das RobMsg-Protokoll besteht aus einem Satz von Routinen, die im Source vorliegen und zum Programm, das sie verwendet gelinkt werden, sowie den zugehörigen Include-Files. Momentan werden diese Routinen zu STEREO, `radio_comm`, `os9_messaging`, `vision` auf der VISION-E7 sowie einem Testprogramm unter MARS gelinkt. Eine gute Kapselung der Daten und die Modularität der Routinen sollten eine Portierung auf andere Systeme bzw. objektorientierte Sprachen problemlos möglich machen.

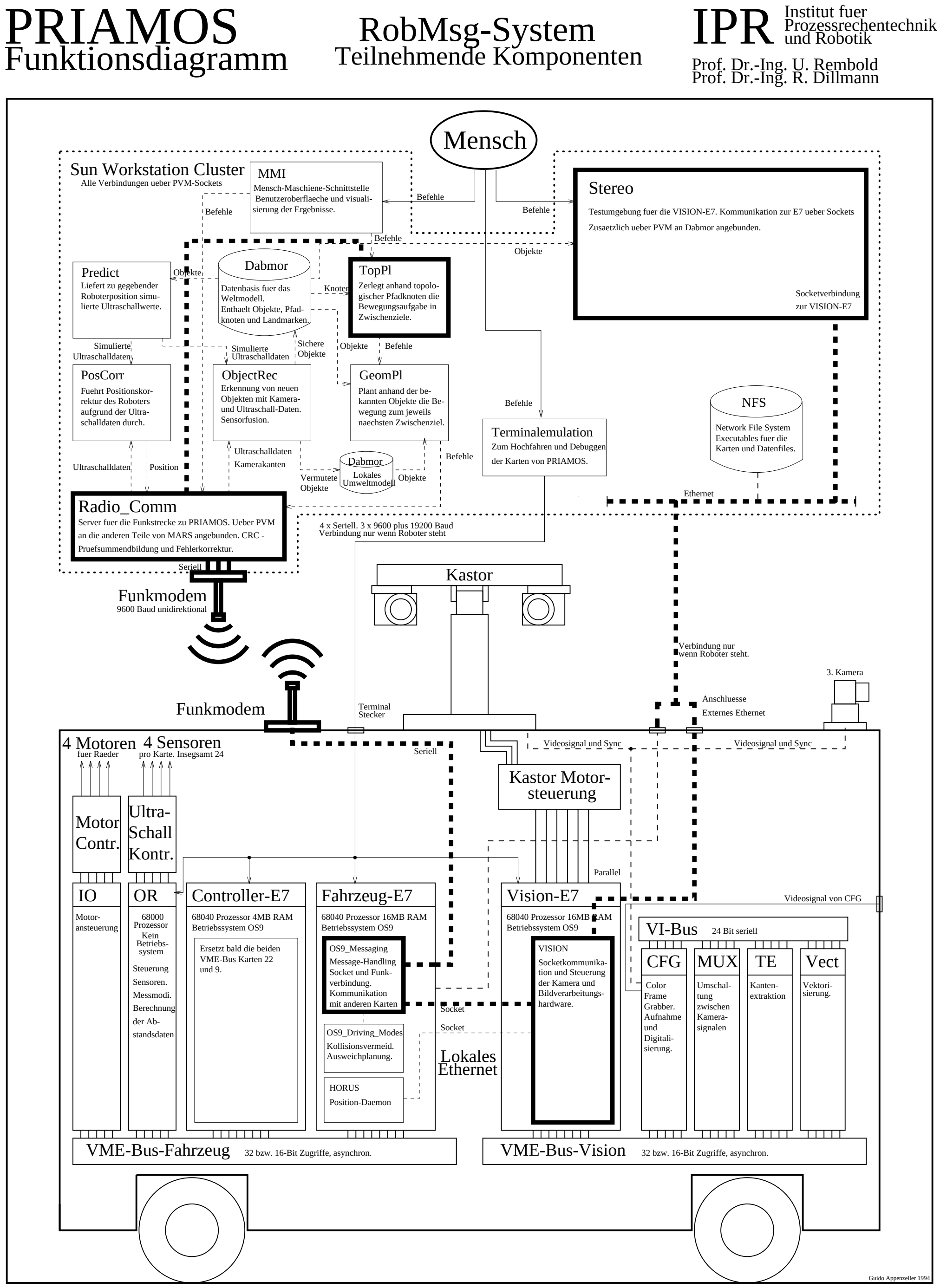

Figure 1: Ablaufplan des RobMsg-Systems

2 Benutzen des Systems

2.1 Einbinden des Systems

Um von einem Programm bzw. Programmteil aus das RobMsg-System benutzen zu können, sind folgende Schritte notwendig:

- Die Files `robmsg.h` und `robmsg.itf` müssen per `#include` eingebunden werden.
- Das Files `robmsg.c` muß kompiliert und hinzugelinkt werden. Hierbei muß beim Kompilieren das entsprechendes Symbol `RM_ENV_Name` aus `robmsg.itf` gesetzt sein, da sonst die systemspezifischen Teile von `robmsg.c` nicht richtig übersetzt werden.
- Sofern nötig, muß der Komunikationskanal initialisiert werden. Z.B. muß ein Socket vorher geöffnet werden, oder ein PVM-Task muß sich beim PVM-Dämon anmelden.

Das Setzen des Symbols geschieht über eine entsprechende Compileroption, die im Makefile bzw. Imakefile angegeben wird. Ein `#define` würde ein Kopieren des Quellkodes erfordern und ist wenig sinnvoll.

2.2 Neue Systeme unter PVM

Neue Systeme unter PVM können eingebunden werden, ohne Änderungen im Source vornehmen zu müssen. Hierzu trägt man den Namen des Programms als einen der Namen in `robmsg.itf` ein, also z.B.:

```
#define RM_NAME_MARS_5 ``ProgName´´
```

und setzt dann beim kompilieren das entsprechende Environment-Symbol, hier also in diesem Fall `RM_ENV_MARS_5`. Nach einem Neuübersetzen von `radio_comm` kann das Programm am RobMsg-System teilnehmen.

2.3 Behandeln von Nachrichten

Eine Nachricht im RobMsg-System entspricht der `ROBMSG` Struktur. Auf diese Struktur sollte **NIE** direkt sondern **NUR** über die entsprechenden Routinen zugegriffen werden.

Um eine Nachricht zu verschicken, wird diese erst mit dem Befehl `NewRobMsg()` angelegt. Hierbei können als Parameter gleich der benötigte Speicher und der Empfänger der Message angegeben werden. Mit `GetRobMsgData()` kann nun ein Zeiger auf den Speicherbereich geholt werden und Daten in die Message geschrieben werden.

Mit `SendRobMsg()` wird die Nachricht abgeschickt. Der Speicher der Nachricht wird dabei freigegeben.

Um Nachrichten zu empfangen, wird `ReceiveRobMsg()` (nicht-blockierend) bzw. `WaitRobMsg()` (blockierend) verwendet. Aus der erhaltenen Nachricht kann nun mit Hilfe von `GetRobMsgFrom()` der Empfänger festgestellt werden und mit `GetRobMsgData()` der Zeiger auf die Daten geholt werden.

Nach der Bearbeitung wird die Message entweder mit `ReplyRobMsg()` zurückgeschickt oder mit `DelRobMsg()` gelöscht. Beide Befehle geben auch den Speicher der RobMsg frei.

2.4 Die ROBMSG-Struktur

Die ROBMSG Struktur enthält im Wesentlichen fünf Elemente:

- Den Absender der Nachricht
- Den Empfänger der Nachricht
- Den Typ der Nachricht, d.h. das Kommando das die Nachricht enthält
- Einen Zeiger auf den Datenbereich der Nachricht
- Die Länge des Datenbereichs der Nachricht.

Das Anlegen, Verändern und Löschen dieser Struktur sollte **NUR** über die aufgeführten Routinen geschehen. Bedeutung und Namen der einzelnen Komponenten können sich jederzeit ändern, die Routienenschnittstelle wird jedoch gleich bleiben.

Für die Adressen der beteiligten Hosts und für die Kommandos sind per `#define` symbolische Namen in `robmsg.itf` festgelegt. Alle Host tragen den Namen `RM_HOST_XXXX` die Kommandos den Namen `RM_CMD_XXXX`.

2.5 Ein kurzes Beispiel

```
/*
 *  Kleines Testprogramm das HELLO-Message an VISION schickt.
 *
 */

#include "robmsh.itf"
#include "robmsg.h"

main()
{
  ROBMSG* message,answer;

/*
 *  Nun zuerst was fuer den startup noetig ist z.B. bei PVM anmelden.
 */

  ...Systemspezifisch...

/*
 *  Message an VISION-E7 mit Kommando HELLO und Datenbereich Laenge 20
 */

  message = NewRobMsg(RM_HOST_VISION,RM_CMD_HELLO,20);

  if( message != NULL )
    {
      strcpy(GetRobMsgData(message),"Hallo Vision!"); /* Daten eintragen */

      if( SendRobMsg(message) == TRUE )             /* RobMsg Abschicken */
{
 /*
  *  Auf eine Antwort warten und diese ausgeben. Im Datenteil auf
  *  die Antwort einer Hello-Nachricht steht ein String, diesen
  *  ausgeben. Am ende die empfangene ROBMSG loeschen.
  */

  answer = WaitRobMsg();
  printf("Antwort:%s\n",GetRobMsgData(answer));
  DelRobMsg(answer);
}
    }
}
```

2.6 Werkzeuge zur Fehlersuche

Innerhalb des RobMsg-Systems gibt es zwei wesentliche Werkzeuge um Fehler aufzusp"uren. Zum einen erm"oglicht das setzen einer Reihe von Symbolen mit `#define` die Ausgabe von Statusmeldungen, zum Anderen ist es m"oglich mit dem Programm `test_robmsg`, welches unter MARS laueft die Verbindungen auszutesten.

2.6.1 Debug-Symbole

Am Anfang von `robmsg.c` stehen drei Debug-Symbole die gesetzt oder nicht gesetzt werden k"onnen:

1. `ROBMSG_DEBUG` gibt eine grosse Menge von Statusmeldungen aus, z.B. wie Nachrichten angelegt, gel"oscht, empfangen, gesendet etc. werden. Es ist nur f"ur die unmittelbare Fehlersuche geeignet.

2. `HANDLE_DEBUG` gibt fuer jede Message die ein System durchl"auft eine Status meldung aus, aus der ersichtlich ist ob die Message das System ordnungsgem"aßdurchlaufen hat.

3. `ERROR_DEBUG` gibt in bestimmten Fehlersituationen Meldungen aus.

Normalerweise ist lediglich `ERROR_DEBUG` gesetzt.

2.6.2 Das Debugtool robmsg_debug

`robmsg_debug` ist eine Testsuite f"ur das RobMsg-System. Sie l"auft unter MARS und testet verschiedene Eigenschaften des RobMsg-Systems und der Funkstrecke.

Es wird zuerst getestet welche anderen Hosts am RobMsg-System teilnehmen. Ein zweiter Test testet die Geschwindigkeit der Funkstrecke. Ein dritter Test ueberpr"uft die Langzeitverl"aßlichkeit der Funkstrecke.

3 Internas des Systems

Dieser Teil der Dokumentation beschreibt den internen Aufbau des Systems. Er ist deshalb weniger für Benutzer der Programmierschnittstelle, sondern für Programmierer, die in das System eingreifen wollen gedacht.

3.1 Funktionsweise

Jede Message, die ein System erreicht, wird an die Routine `HandleMessage()` übergeben. Diese Routine überprüft nun, ob diese Message an das System selbst geht oder an ein anderes System weitergeleitet werden muß.

Messages, die an das System selbst gehen werden in die lokale Message-Queue eingehängt und warten dort darauf bis sie mit `ReceiveRobMsg()` abgehohlt werden.

Messages an andere Systeme werden an `PutRobMsg()` übergeben. `PutRobMsg()` wird durch eingefügte `#ifdef`-Befehle auf jedem System anders übersetzt. Es enthält zusammen mit `GetRobMsg()` fast den gesamten Systemspezifischen Teil von `robmsg.c` . Die eigentlichen Sende- bzw. Empfangs-Routinen sind nur noch verbindugsspezifisch.

3.2 Ablaufplan des Systems

3.3 Verbindungstypen

3.3.1 Sockets

Wenn RobMessages über Sockets verschickt werden wird von `PutRobMsg()` in die Routine `WriteRobMsgToSocket()` gesprungen. Der Socket in den geschrieben wird wird als Parameter mit übergeben. Die Routine `WriteRobMsgToSocket()` ist nicht mehr Systemabhängig. In den Socket wird zuerst die Bytefolge '*' und Nullbyte geschrieben. Dies dient zur Unterscheidung zwischen anderen Daten die auf dem Socket transportiert werden und RobMessages. Über das globale Flag `SOCKET_ACKNOWLEDGE` kann eingestellt werden ob auf dem Socket eine Bestätigung stattfinden soll oder nicht. Einschalten der Bestätigungen führt zu langsameren Nachrichtenlaufzeiten.

RobMessages werden mit `ReadRobMessageFromSocket()` empfangen. Diese Routine erwartet, daß bereits zuvor die beiden ersten Bytes vom Socket gelesen worden sind.

Es wird bei aufruf der Routinen immer davon ausgegangen das der angegebene Socket bereits geöffnet wurde. Socket-Routinen stehen momentanauf der Fahrzeug-E7, in VISION und in STEREO zur verfügung.

3.3.2 Funkstrecke

Auf der Funkstrecke existierrte bereits ein Funkprotokoll auf das aufgebaut werden konnte. Die RobMessages werden indieses Protokoll verpackt und ueber die Funkstrecke geschickt. Die Porbleme des Funkprotokolls selbst werden weiter unten ausführlich behandelt.

Die Routinen für das Verschicken bzw. Empfangen sind `WriteRobMsgRadio()` bzw. `ReadRobMsgRadio()`. Jede ROBMSG entspricht einer Radio-Message. Der Datenteil der Radio-Message enthält die ROBMSG und den Datenteil der ROBMSG.

Die Funkstreckenroutinen werden in `radio_comm` und auf der Fahrzeug-E7 eingebunden.

3.3.3 PVM

Unter PVM kommunizieren die Prozesse ueber PVM-Eigene Messages. Als Router fuer die Programme in MARS fungiert `radio_comm`. Alle Nachrichten werden erst an `radio_comm` und von dort aus an den Empfänger geschickt. Die TID wird von `PutRobMsg()` ermittelt und den ausführenden Routinen `WriteRobMsgToPVM()` bzw. `ReadRobMsgFromPVM()` als Parameter übergeben.

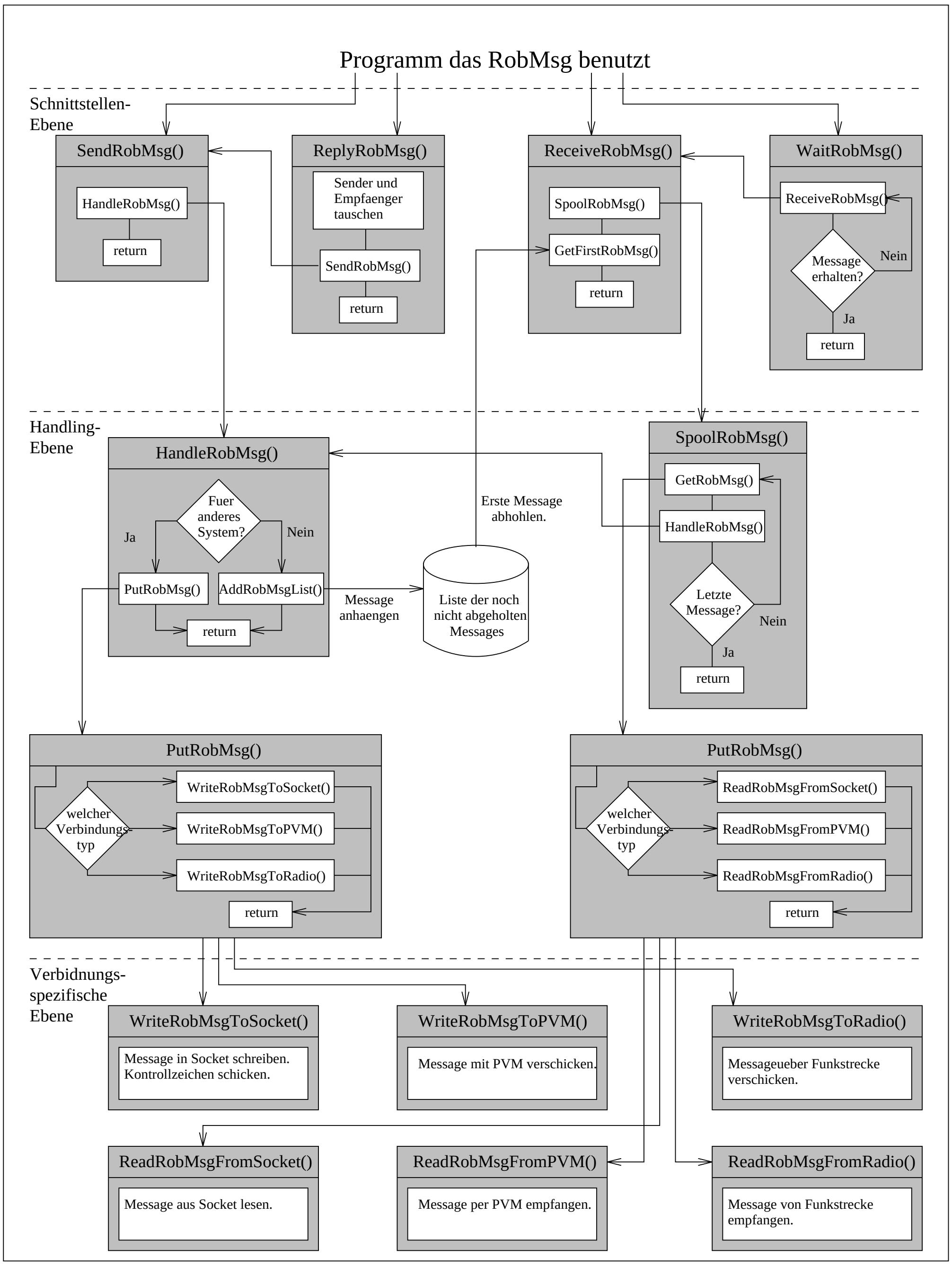

Figure 2: Ablaufplan des RobMsg-Systems

4 Die RadioComm Fehlererkennung

4.1 Problemstellung

Auf der Funkstrecke zwischen Fahrzeug E-7 und Workstation wird ein Funkmodem eingesetzt, das nur unidirektional und ohne jeden Handshake übetragen kann.

Bei dem Betrieb des Modems stellten sich zwei Fehlerquellen heraus. Zum einen führt der fehlende Handshake zwischen Modem und Rechner zu Überläufen des 64-Byte Puffers des Modems. Während bei unidirektionalem Betrieb mit ausreichend schnellem Leeren des Puffers auf der Empfängerseite dieses Risiko recht klein ist (da die Serielle Übertragungsrate Sun-Modem gleich der des Modems), treten Pufferüberläufe bei einer Auslastung in beide Richtungen häufig auf. Versuche ergaben, daß ohne Korrekturprotokoll eine Pause von mindestens eine Sekunde gelassen werden mußte, um eine geringe Fehlerrate zu erreichen.

Die zweite Fehlerquelle sind Übertragungsfehler des Modems. Diese Fehler lassen sich mit Hilfe der CRC-Prüfsumme problemlos feststellen. Sie machen das Einbauen einer Reihe von zusätzlichen Sicherheitsvorkehrungen notwedig.

Das Funkprotokoll von Christian Abeln stellte auf der Funkstrecke lediglich eine funktionierende CRC-Prüfsumme zur Verfügung. Der Acknowledgebetrieb führte bei einer bidirektionalen Auslastung zu einem Zusammenbrechen der Kommunikation. Ohne Acknowledge wurden fehlerhafte Messages zwar erkannt und verworfen, bei mehreren Nachrichten die kurz hintereinander geschickt werden kommt jedoch nur die erste an. Die restlichen werden durch Pufferüberläufe gestört und nach der CRC-Prüfung gelöscht.

Die oben geschilderten Probleme wurden durch das Einführen von Warteschlagen, eines Sendetokens und eines Fehlerkorrekturprotokolls behoben.

4.2 Das Tokenprinziep

Die Schnittstellen zu den Funkmodems werden mit der selben Baudrate wie das Modem betrieben. Solange das Modem senden kann, d.h. solange eine Verbindung existiert und es nicht gerade in die andere Richtung betrieben wird, k"onnen keine Puffer"uberl"aufe auftreten.

Um sicherzustellen das immer nur in eine Richtung gesendet wird, wird zwischenden Beiden Seiten ein Sendetoken hin- und hergereicht. Nur die Seite die den Token hat darf auch senden. Der Token besteht aus den Buchstaben 'g' und 'o'.

Um gleichzeitig noch sicherzustellen das ein `SendMessage()`-Aufruf sofort zur"uckkommt werden Nachrichten in dieser Funktion nur in eine interne Warteschlange eingereiht. Sobald `GetMessage()` aufgerufen wird "uberpr"uft ob der Token oder eine Message empfangen wurde. Sobald der Token empfangen wurde wird eine Message (sofern vorhanden) aus der Schlange geschickt und danach der Token an die Gegenseite geschickt. `GetMessage()` bearbeitet empfangene Nachrichten sofort.

Das Tokensystem f"uhrt wahrscheinlich zu einer leichten Verlangsamung der Funkverbindung. Da keine Messung der Geschwindigkeit vorher m"oglich war ist ein Vergleich nicht m"oglich. Momentan k"onnen ca. f"unf Messages pro Sekunde (von L"ange 0) versandt werden.

Wenn `radio_comm` f"ur 5 Sekunden keinen Token vom Roboter zur"uckbekommt geht es davon aus das der Token verlorengegangen ist und sendet erneut den Token.

4.3 Die Fehlerkorrektur

Eine echte Fehlerkorrektur findet momentan nicht statt, die Fehlerrate wird jedoch bereits durch das Tokensystem sehr niedrig gehalten.

5 Liste der Routinen

5.1 Routinen um ROBMSGs zu verändern

NewRobMsg() - Neue ROBMSG anlegen

Aufruf:

```
ROBMSG *NewRobMsg(to,command,length)
```

Parameter:

`char to`	Adresse des Hosts an den Message gehen soll
`char command`	Typ bzw. Kommando der Message
`int length`	Länge des Datenbereichs in Byte

Rückgabewert:

Pointer auf die ROBMSG-Struktur oder NULL wenn nicht genug Speicher frei ist.

Beschreibung:

Alloziert Speicher für eine RobMsg-Struktur und, sofern nötig, ihren Datenbereich. Das Absenderfeld der ROBMSG wird automatisch richtig gesetzt. Neue ROBMSGs sollten **NUR** mit dieser Funktion angelegt werden.

DelRobMsg() - ROBMSG Löschen

Aufruf:

```
int DelRobMsg(ROBMSG *msg);
```

Parameter:

`ROBMSG *msg`	Pointer auf eine ROBMSG

Rückgabewert:

TRUE bei erfolg, FALSE falls ein Nullpointer übergeben wurde.

Beschreibung:

Diese Funktion löscht ROBMSGs. Jede einmal angelegte oder empfangene ROBMSG muß entweder durch Abschicken (`ReplyRobMsg()`, `SendRobMsg()` oder durch diese Funktion wieder gelöscht werden.

ChangeRobMsgLength() - Länge Datenbereich ändern

Aufruf:

```
int ChangeRobMsgLength(ROBMSG *msg,int length);
```

Parameter:

`ROBMSG *msg`	Pointer auf eine ROBMSG
`int length`	Neue gewünschte Länge der ROBMSG

Rückgabewert:

TRUE bzw FALSE je nachdem ob gewünschte Länge erzeugt werden konnte.

Beschreibung:

Die Länge des Datenbereichs der ROBMSG wird geändert. Hierzu wird erst der bisherige Speicher der ROBMSG freigegeben und dann versucht, neuen zu allocieren. FALSE wird zureuckgemeldet wenn nicht genug Speicher frei ist oder ein Nullpointer übergeben wurde.

ChangeRobMsgCommand() - Command der ROBMSG ändern

Aufruf:

```
int ChangeRobMsgCommand(ROBMSG *message, char Command);
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG
`char Command`	Neues Kommando wie in `robmsg.itf` definiert

Rückgabewert:

TRUE bzw FALSE je nachdem ob Kommando geändert werden konnte.

Beschreibung:

Das Kommando der ROBMSG wird geändert. Um Datenkapselung zu garantieren, sollte immer diese Routine verwendet und nie direkt auf die Struktur zugegriffen werden.

ChangeRobMsgTo() - Empfänger der ROBMSG ändern

Aufruf:

```
int ChangeRobMsgTo(ROBMSG *message, char To);
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG
`char To`	Neuer Empfänger der ROBMSG wie in `robmsg.itf`

Rückgabewert:

TRUE bzw FALSE je nachdem ob Kommando geändert werden konnte.

Beschreibung:

Der Empfänger der ROBMSG wird geändert. Um Datenkapselung zu garantieren, sollte immer diese Routine verwendet und nie direkt auf die Struktur zugegriffen werden.

GetRobMsgData() - Zeiger auf Datenbereich der ROBMSG hohlen

Aufruf:

```
void *GetRobMsgData(ROBMSG *message);
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

Zeiger auf den Datenbereich der ROBMSG

Beschreibung:

Holt den Zeiger auf den Datenbereich der ROBMSG. In diesen Speicherbereich können dann die Daten geschrieben werden, die mit der ROBMSG verschickt werden sollen. Die Länge des Bereichs läßt sich mit `GetRobMsgLength()` feststellen.

GetRobMsgLength() - Länge des Datenbereichs der ROBMSG hohlen

Aufruf:

```
int GetRobMsgLength(ROBMSG *message);
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

Länge des Datenbereichs in Bytes oder 0 bei Fehler

Beschreibung:

Holt die Länge des Datenbereichs.

GetRobMsgCommand() - Kommando der ROBMSG hohlen

Aufruf:

```
char GetRobMsgCommand(ROBMSG *message)
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

Kommando der ROBMSG wie in `robmsg.itf` definiert bzw. 0 wenn ein Fehler auftritt.

Beschreibung:

Holt Kommando der ROBMSG.

GetRobMsgTo() - Empfänger der ROBMSG hoheln

Aufruf:

```
char GetRobMsgTo(ROBMSG *message)
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

Empfänger der ROBMSG, wie in `robmsg.itf` definiert bzw. 0, wenn ein Fehler auftritt.

Beschreibung:

Hohlt Empfänger der ROBMSG.

GetRobMsgFrom() - Absender der ROBMSG hohlen

Aufruf:

```
char GetRobMsgFrom(ROBMSG *message)
```

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

Sender der ROBMSG, wie in `robmsg.itf` definiert bzw. 0, wenn ein Fehler auftritt.

Beschreibung:

Hohlt Sender der ROBMSG.

5.2 Routinen um ROBMSGs auszutauschen

SendRobMsg() - Absenden einer ROBMSG

Aufruf:

`int SendRobMsg(ROBMSG *message)`

Parameter:

`ROBMSG *message`	Pointer auf eine ROBMSG

Rückgabewert:

TRUE oder FALSE je nachdem ob die Nachricht verschickt werden konnte oder nicht. TRUE garantiert nicht, daß die Nachricht auch wirklich angekommen ist, sondern lediglich, daß unser System es noch geschafft hat, die Nachricht abzusenden.

Beschreibung:

Eine vorher angelegte und gefüllte Nachricht wird an einen anderen Teilnehmer im ROBMSG-System geschickt. Vorher müssen das Absenderfeld, das Empfängerfeld und das Kommando-Feld richtig gesetzt werden. Diese Routine kommt immer sofort zurück. Das eigentliche Absenden der Nachricht kann (z.B. bei der Funk-Strecke) erst einige Sekunden später erfolgen.

ReceiveRobMsg() - Nichtblockierendes Empfangen einer ROBMSG

Aufruf:

`ROBMSG *ReceiveRobMsg(void)`

Parameter:

`keine`

Rückgabewert:

Wenn eine ROBMSG für uns anliegt ein Zeiger auf die ROBMSG sonst NULL

Beschreibung:

Dies ist die nichtblockierende Methode Messages, die an uns gerichtet sind abzuhohlen. Die Messages, die bei unserem System ankommen werden erst in einer Warteschlange zwischengespeichert, bis sie abgehohlt werden. Mit `ReceiveRobMsg()` bzw. `WaitRobMsg()` wird lediglich das erste Element der Schlange gehohlt. Es können nur

Nachrichten empfangen werden, die auch wirklich an uns gerichtet sind. Messages die das System lediglich routed, tauchen hier nicht auf. Wenn keine ROBMSG für uns anliegt, kommt die Routine sofort zurück.

WaitRobMsg() - Blockierendes Empfangen einer ROBMSG

Aufruf:

`ROBMSG *WaitRobMsg(void)`

Parameter:

`keine`

Rückgabewert:

Zeiger auf die ROBMSG

Beschreibung:

Blockierendes empfangen einer ROBMSG. Diese Routine wartet im Gegensatz zu `GetRobMsg()`, bis eine ROBMSG empfangen wurde. In einigen Systemen (z.B. VISION) ist diese Routine nicht anwendbar, da dort eine Routine ausserhalb des ROBMSG-Packets für das Message-Abhohlen zuständig ist. Hier muß diese externe Routine zusammen mit `ReceiveRobMsg()` in einer Schleife aufgerufen werden.

ReplyRobMsg() - Zurückschicken einer ROBMSG

Aufruf:

`int ReplyRobMsg(ROBMSG *message)`

Parameter:

`ROBMSG *message` Pointer auf eine ROBMSG

Rückgabewert:

TRUE oder FALSE wie bei `SendRobMsg`

Beschreibung:

Schickt eine ROBMSG die mit `ReceiveRobMsg()` empfangen wurde an den Absender zurück. Alle Daten werden dabei mit zurückgeschickt. Vorher kann bei Bedarf der Datanbereich noch mit `ChangeRobMsgLength(msg,0)` geändert werden.