System Architecture Specification

**(TINF11D, SWE I Praxisprojekt 2011/2012)**

|  |  |
| --- | --- |
| **Projekt:** | **MultiCastor 3.0** |
|  |  |
| **Auftraggeber:** | **Rentschler & Stuckert**  **Rotebühlplatz 41**  **DHBW Stuttgart** |
|  |  |
| **Auftragnehmer:** | **TINF11D – Team 4**  **Nick Herrmannsdörfer**  **Stefan Heßler**  **Erwin Stamm**  **Kai Brennenstuhl**  **Patrick Robinson** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Datum | Autor | | Kommentar |
| 0.1 | 09.11.2012 | Erwin Stamm | Dokument angelegt | |
| 0.2 | 10.11.2012 | Patrick Robinson | Systemumgebung, Klassendiagramm | |
| 0.3 | 12.11.2012 | Patrick Robinson | Nicht-funktionale Anforderungen | |
| 0.4 | 13.11.2012 | Erwin Stamm | GMRP-Spezifikationen | |

Inhaltsverzeichnis

[1. Einführung 5](#_Toc340609307)

[1. Integration eines neuen Protokolls – GMRP 6](#_Toc340609308)

[1.1. Multicast Mac-Adresse 6](#_Toc340609309)

[1.2. Aufabau des GMR-Protokolls 6](#_Toc340609310)

[1.2.1. Nachrichtentypen beim GMR-Protokoll 6](#_Toc340609311)

[1.2.2. Beschreibung der Zustandsautomaten 7](#_Toc340609312)

[1.1.1. Kommunikation – Verbindung vom Empfänger zu Sender herstellen 7](#_Toc340609313)

[1.2.3. Kommunikation – Verbindung vom Empfänger zum Sender trennen 8](#_Toc340609314)

[1.3. Systemdesign und Programmstruktur 8](#_Toc340609315)

[1.3.1. Zustandsautomat – Registrar state machine 8](#_Toc340609316)

[1.3.2. Zustandsautomat – Applicant state machine 11](#_Toc340609317)

[1.3.3. Zustandsautomat - LeaveAll state machine 14](#_Toc340609318)

[1.3.4. Zustandsautomat - Periodic Transmission state machine 15](#_Toc340609319)

[1.4. GMRP-Paket 17](#_Toc340609320)

[1.4.1. GARP PDU Struktur 17](#_Toc340609321)

[1.4.2. GARP Nachricht Struktur 17](#_Toc340609322)

[1.4.3. Attribut Struktur 17](#_Toc340609323)

[2. Referenzen 18](#_Toc340609324)

[3. Produktfunktionen 19](#_Toc340609325)

[4. Anforderungen 20](#_Toc340609326)

[5. Produktcharakteristiken 21](#_Toc340609327)

[5.1. Systemumgebung 21](#_Toc340609328)

[5.1.1. Hardwareumgebung 21](#_Toc340609329)

[5.1.2. Softwareumgebung 21](#_Toc340609330)

[5.2. Nicht-funktionale Anforderungen 22](#_Toc340609331)

[5.2.1. LL10/ Fehlerhandling 22](#_Toc340609332)

[5.2.2. LL20/ Effiziente Nutzung 22](#_Toc340609333)

[5.2.3. LL30/ Benutzerfreundlichkeit 22](#_Toc340609334)

[6. Referenzen 23](#_Toc340609335)

**Abbildungsverzeichnis:**

[Abbildung 1: Übersicht über Systemumgebung 8](#_Toc340523214)

[Abbildung 2: Klassendiagramm 10](#_Toc340523215)

# Einführung

Das vorliegende Dokument beschreibt die architekturspezifischen Änderungen, von Multicastor Version 2.0 auf 3.0. Im Folgenden wird Multicastor mit MC oder MCaster abgekürzt und bei einer nicht Spezifizierung der Versionsnummer, ist von der Version 3.0 auszugehen. Bei der Darstellung von Binärzahlen wird die LBS(Least Signifikant Bit First) Schreibweise verwendet. Weiterhin verweist die Nutzung der ersten Person-Plural Form, auf die Gruppenmitglieder des MC-Teams.

Der MC baut auf dem Model View Controller Architekturmuster auf. Aufbauend auf der Version 2.0 gibt es weiterhin eine zusätzliche Aufteilung in interfaces, lang(language), data und mmrp. Die einzelnen Komponenten sind in Packages unterteilt mit der jeweiligen Bezeichnung. Zusätzlich zu den vorliegenden Komponenten wird der MC um die Komponente GMRP erweitert, diese wiederum entspricht dem Aufbau der MMRP Komponente.

GUI

Die Änderungen an der GUI sollen so gering wie möglich gehalten werden, um einen Wiedererkennungswert zu gewährleisten. Trotzdem werden Änderungen vorgenommen um Fehlinformationen zu vermeiden und einen genaueren Einblick in die Einzelnen Multicaststeams zu ermöglichen. Weiterhin soll jede Option auch nur an einer Stelle innerhalb der GUI einstellbar sein. Um die GUI übersichtlich zu gestalten und damit die Verwirrung des Benutzers zu vermindern. Bestehende Fehler wie vor allem die Inkompatibilität zu Java 7 sollen behoben werden.

# Integration eines neuen Protokolls – GMRP

In diesem Abschnitt wird das GMR-Protokoll dargestellt. Es wird erklärt, welche Aufgaben das Protokoll zu erledigen hat, wie die Kommunikation abläuft, wie der Aufbau der Zustandsautomaten und des GMRP-Paketes aussieht. Bei der Implementierung wird der Aufbau des Implementieren MMR-Protokoll verwendet und gemäß der GMRP-Spezifikationen abgeändert.

## Multicast Mac-Adresse

Das letzte Bit des ersten Bytes der Mac-Adresse wird genutzt um zwischen einer Multicast und einer Unicast-Adresse zu unterscheiden.

Schaubild einfügen.

## Aufabau des GMR-Protokolls

### Nachrichtentypen beim GMR-Protokoll

Es gibt 6 Nachrichtentypen, die im GMRP definiert sind. Jeder Nachrichtentyp erfüllt seine eigene Funktion.

1. JoinIn (*Applicant* deklariert und registriert)
2. In (*Applicant* ***nicht*** deklariert und registriert)
3. JoinEmpty (*Applicant* deklariert und **nicht** registriert)
4. Empty (*Applicant* **nicht**deklariert und **nicht** registriert)
5. Leave (*Applicant* deregistrieren)
6. LeaveAll (Alle *Applicants* deregistrieren)

### Beschreibung der Zustandsautomaten

***Registrar state machine***: Die *Registrar state machine* verwaltet die registrierten MAC-Adressen. Pro Multicast-Strom, der gesendet wird, benötigt man eine *Registrar state machine.* Ein Switch, der einen Multicast Strom nur empfängt, benötigt keine *Registrar state machin* zu initialisieren.

***Applicant state machine:*** Die *Applicant state machine* wird genutzt, um die Kommunikation im Netzwerk herzustellen. Die *Applicant state machine* versendet Nachrichten ins Netzwerk und stellt die Kommunikation zwischen *Applicant state machine* und/oder *Registrar state machine* her. Pro Multicast-Strom, der gesendet bzw. empfangen wird, benötigt man eine *Applicant state machine.*

***Application-Only:*** Als *Application-Only* bezeichnet man eine *Applicant state machine,* die nur Multicastströme empfängt, sie jedoch nicht sendet.

***LeaveAll state machine:*** Die *LeaveAll state machine* versendet in bestimmten Zeitpunkten LeaveAll Nachrichten, um unbenutzte Multicastströme zu entfernen. Jeder Switch benötigt nur eine *LeaveAll state machine.*

***PeriodicTransmission state machine:*** Eine *PeriodicTransmission state machine* sendet an die dazu gehörigen *Applicant state machine* Nachrichten, dass die *Applicant state machine* aktive JoinIN Nachrichten verschicken soll. Dies soll einerseits zur Überprüfung dienen, ob der Multicaststrom vorhanden ist und andererseits ist bei Netzwerkänderungen garantiert, dass die Verbindung schnell hergestellt wird.

### Kommunikation – Verbindung vom Empfänger zu Sender herstellen

1. *Application-Only* vom Switch 1 schickt eine JoinIn Nachricht an die *Applicant state machine* am Port 2 in Switch 1.
2. Die *Registrar state machine* am Port 2 registriert die MAC-Adresse vom Host und teilt der *Applicante state machine* mit, dass eine MAC-Adresse registriert wurde.
3. Die *Applicant state machine* sendet daraufhin eine JoinIn Nachricht an den *Application-Only* zurück.
4. Der *Application-Only* sendet als Bestätigung eine JoinIn Nachricht zurück.
5. Die *Applicant state machine* führt dieselbe Art von Bestätigung nochmals aus. Die erste Verbindung wurde erfolgreich hergestellt.
6. Um Switch 1 mit Switch 2 zu verbinden, müssen die zwei *Applicant state machines* ebenfalls eine Verbindung herstellen. Die Verbindungsherstellung funktioniert genauso wie in Schritt 1 bis 5 beschrieben.
7. Dieser Vorgang zieht sich durch bis die *Applicant state machine* vom Sender verbunden ist. Damit ist die Verbindung vom Empfänger zum Sender hergestellt.

### Kommunikation – Verbindung vom Empfänger zum Sender trennen

1. *Application-Only* vom Switch 1 schickt eine Leave Nachricht an die *Applicant state machine* am Port 2 in Switch 1.
2. Da keine JoinIn Nachricht vom *Application-Only* mehr kommt, deregistriert die *Registrar state machine die MAC-Adresse.* Daraufhin schickt die *Applicant state machine* ebenfalls eine Leave-Nachricht an Switch 2.
3. Dieser Vorgang wiederholt sich, bis sie beim Sender ankommt. Die Verbindung wurde erfolgreich aufgelöst.

## Systemdesign und Programmstruktur

In diesem Abschnitt werden die einzelnen Zustandsautomaten dargestellt. Es wird erläutert, wie sie aufgebaut sind und wie sie auf bestimmte Events reagieren. Des Weiteren wird der Aufbau des GMRP-Pakets dargestellt. Hier wird verdeutlicht, wie sich der Aufbau einer JoinIn Nachricht von beispielsweise einer JoinEmpty Nachricht unterscheidet.

### Zustandsautomat – Registrar state machine

**Zustandsautomat:**

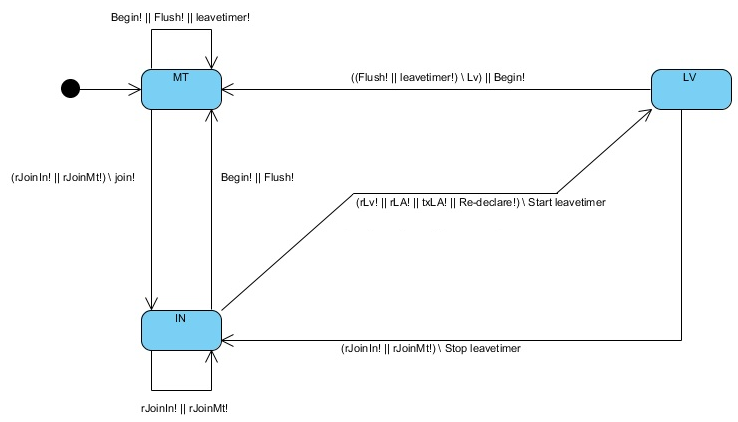


Abbildung 15: Zustandsautomat Registar state machine

**Zustandsübergangstabelle:**

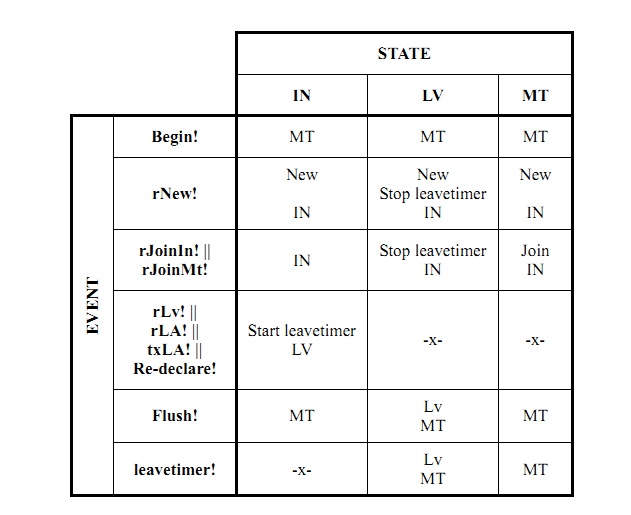


Abbildung 16: Registrar state machine[[1]](#footnote-1)

**Zustände:**

|  |  |
| --- | --- |
| Zustand | Bedeutung |
| IN | MAC-Adresse wurde gespeichert |
| LV | Registrar hat eine Leave Nachricht erhalten. Wenn keine JoinIn Nachricht kommt, wechselt der Automat in den MT Zustand |
| MT | Registrar ist leer |

**Events:**

|  |  |
| --- | --- |
| Event | Bedeutung |
| Begin! | Event zum Starten und initialisieren des Zustandsautomaten. |
| rNew! | Event zum schnellen Registrieren von Attributen. (Findet keine Verwendung in GMRP.) |
| rJoinIn! | Empfang einer JoinIn-Nachricht über das Netzwerk |
| rJoinMt! | Emfpang einer JoinMT-Nachricht über das Netzwerk. (Wird von der Applicant state machine generiert. Der Unterschied zur JoinIn-Nachricht wird dort erklärt) |
| rLv! | Empfang einer Leave-Nachricht über das Netwerk. Dient zur Deregistrierung |
| rLA! | Empfang einer LeaveAll-Nachricht über das Netzwerk. Zur Deregistrierung durch den garbage collector. |
| txLA! | Event für die Übertragungsmöglichkeit und einer LeaveAll-Nachricht tritt ein. Hier keine größere Änderung als bei einem normalen rLA!-Event |
| Re-declare! | Event, das einen Port-Rollen-Wechsel anzeigt. (Von Designated-Port zu Root-Port oder Alternate-Port |
| Flush! | Event, das einen Port-Rollen-Wechsel anzeigt. (Von Root-Port oder Alternate-Port zu Designated-Port.) |
| leavetimer! | Der leavetimer ist abgelaufen und generiert dieses Event, um anzuzeigen, dass der Time-Out zum Reregistrieren vorbei ist. |

**Actions:**

|  |  |
| --- | --- |
| Action | Bedeutung |
| New | Im GMRP wird die Action nie ausgeführt. |
| Join | Sendet eine Join Nachricht an die anknüpfende Applicant state machine. |
| Lv | Sendet eine Leave Nachricht an die anknüpfende Applicant state machine |
| Start leavetimer | Startet den Timer, der definiert, wie lange ein Empfänger Zeit hat sich zu registrieren, nachdem die Verbindung getrennt wurde. |
| Stop leavetimer | Der Timer ist abgelaufen und die regestrierte MAC-Adresse wird deregestriert. |

### Zustandsautomat – Applicant state machine

**Zustandsautomat:**

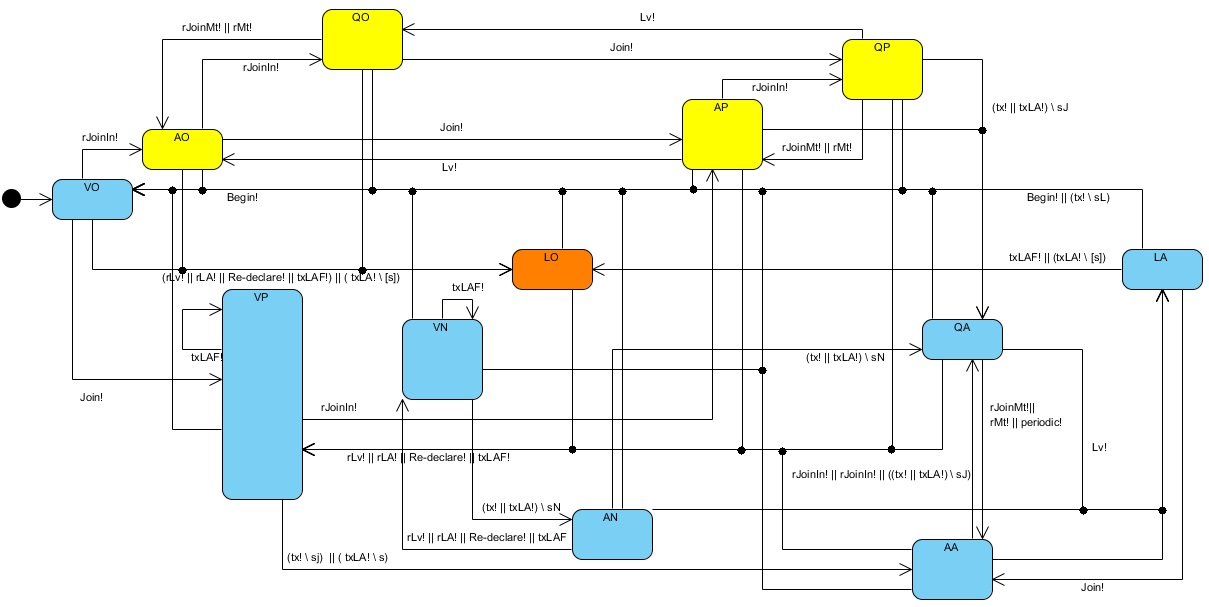


Abbildung 17: Zustandsautomat - Applicant state machine

**Zustandsübergangstabelle:**

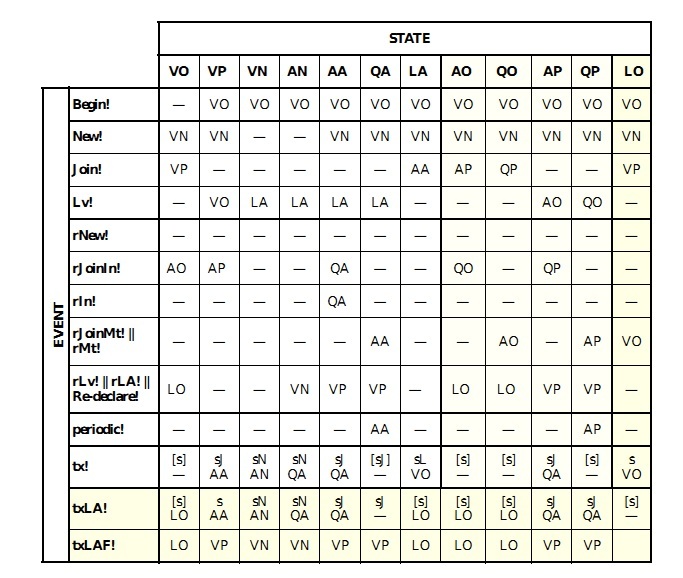


Abbildung 18: Applicant state machine[[2]](#footnote-2)

**Anmerkung:**

* Weiße Zustände stellen einen Application-Only Automaten dar.
* Hellgelbe Zustände fallen weg, wenn es sich um ein Point-to-Poin-Netzwerk handelt.

**Zustände:**

|  |  |
| --- | --- |
| Zustand | Bedeutung |
| VO | *Very anxious Observer* - Zustand, in dem sich der Automat zu Beginn beendet. Dieser wird auch beim Verlassen der Registrierung wieder erreicht. |
| VP | *Very anxious Passive* - Der Automat kommt in den Zustand, da er aufgefordert wurde,  die Adresse zu deklarieren. (Dies geschieht durch die internen Join! -Nachrichent, die beispielsweise der Registrar aussendet.) |
| VN | *Very anxious New* - Nicht weiter beschrieben; GMRP macht keinen Gebrauch des New! -Kommandos, welcher zu diesem Zustand führt |
| AN | *Anxious New* - siehe: VN |
| AA | *Anxious Active* - Die Applikation beendet sich bei der Deklaration der Adresse. Sie hat bereits eine Join-Nachricht versendet und sendet beim nächsten tx!-Event die zweite Join-Nachricht aus. |
| QA | *Quiet Active* - Zustand, in dem die Applikation die Anzahl der nötigen Join-Nachrichten abgesetzt hat. Dies ist der Zustand nach erfolgter Deklaration der Adresse. |
| LA | *Leaving Active* - Die Applikation war bereits bei der Deklaration der Adresse und hat eine interne Lv-Nachricht empfangen. Sie sendet beim auftretenden tx!-Event eine Lv-Nachricht über das Netzwerk aus. |
| AO | *Anxious Observer* - Die Applikation deklariert selbst keine Adressen, hat aber eine JoinIN-Nachricht (durch das Netzwerk) erhalten. |
| QO | *Quiet Observer* - Die Applikation deklariert selbst keine Adressen, hat aber eine JoinIN - Nachricht (durch das Netzwerk) erhalten. |
| AP | *Anxious Passive* - Die Applikation hat eine JoinIN-Nachricht empfangen. Anschließend wird sie aufgefordert, diese Adresse im Netzwerk zu deklarieren. (VO->AO->AP) |
| QP | *Quiet Passive* - Die Applikation hat zwei JoinIN-Nachrichten empfangen, um in den QO - Zustand zu kommen. Nun wird sie intern aufgefordert, selbst die Adresse zu deklarieren.  In der Zwischenzeit sind zwei JoinIN-Nachrichten bei ihr eingegangen, deswegen muss sie selbst keine Nachricht versenden und kann im Zustand QP verbleiben. |
| LO | *Leaving Observer* - Die Applikation deklariert diese Adresse nicht. Sie hat aber eine Lv-, LeaveALL- oder Re-Declare-Nachricht empfangen. Beim nächsten tx!-Event versendet sie eine Empty -Nachricht. |

**Events:**

|  |  |
| --- | --- |
| Event | Bedeutung |
| Begin! | Event zur Initialisierung des Automaten. |
| New! | Event zum schnellen Registrieren von Attributen. (Findet keine Verwendung in GMRP.) |
| Join! | Internes Event; Aufforderung zur Deklaration der Adresse. |
| Lv! | Internes Event; Aufforderung die Deklaration zu deregistrieren. |
| rNew! | Externe New-Message. (Findet keine Verwendung in GMRP.) |
| rJoinIn! | Externes Event; Die Application state machine des verbundenen Switches wurde aufgefordert, die Adresse zu deklarieren. Sie hat eine Join-Nachricht abgesetzt, die sagt, dass ihr eigener Zustandsautomat sich im Zustand IN beendet. (Oder, dass sie eine Applicant-Only state machine ist.) |
| rIn! | Externes Event; der Automat hat den Zustand eines verbundenen Registrars durch einen verbundenen Applicant state machine empfangen. Er befand sich im Zustand IN. |
| rJoinMt! | Externes Event; Join-Nachricht wurde empfangen mit der Information, dass sich der entsprechende Registrar nicht im Zustand IN befindet. |
| rMt! | Externes Event; Die Nachricht informiert darüber, dass sich der entsprechende Registrar nicht im Zustand IN beendet. |
| rLv! | Externes Event; Die Nachricht informiert darüber, dass sich der entsprechende Registrar nicht im Zustand IN beendet. |
| rLA! | Dieses Event kann sowohl von einer externen als auch von einer internen Quelle stammen. Zeigt an, dass eine LeaveAll-Nachricht empfangen wurde. Eventuelle Deklarationen müssen erneut deklariert werden, um einen Abbau der Deklaration zu vermeiden. |
| Re-declare | Internes Event; Erneutes Deklarieren durch Wechsel der Port-Rolle nötig. |
| periodic! | Internes periodisches Event, das durch die Periodic state machine erzeugt wird. |
| tx! | Event, das anzeigt, dass nun eine Übertragungsmöglichkeit existiert. |
| txLA! | Event für die Übertragungsmöglichkeit mit zeitgleicher LeaveAll-Nachricht. |
| txLAF | Event für die Übertragungsmöglichkeit mit zeitgleicher LeaveAll -Nachricht und fehlendem Platz für weitere Adressen. |

**Actions:**

|  |  |
| --- | --- |
| Action | Bedeutung |
| s | Sendet eine IN- oder Empty-Nachricht in Abhängigkeit der zugehörigen Registrar state machine aus |
| [s] | Sendet eine IN- oder Empty-Nachricht in Abhängigkeit der zugehörigen Registrar state machine aus. Diese Aktion kann ausgeführt werden, wenn sie zu Code-Optimierungszwecken gebraucht wird. |
| sJ | Sendet eine JoinIN- oder JoinMt-Nachricht in Abhängigkeit der zugehörigen Registrar state machine aus |
| [sJ] | Sendet eine JoinIN- oder JoinMt-Nachricht in Abhängigkeit der zugehörigen Registrar state machine aus. Diese Aktion kann ausgeführt werden, wenn sie zu Code-Optimierungszwecken gebraucht wird. |
| sL | Sendet eine Lv-Nachricht aus |
| sN | Wird nicht benötigt, da GMRP New!-Events nicht verwendet |

### Zustandsautomat - LeaveAll state machine

**Zustandsautomat:**

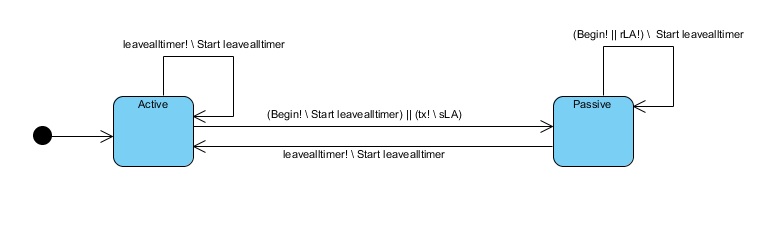


Abbildung 19: Zustandsautomat - LeaveAll state machine

**Zustandsübergangstabelle:**

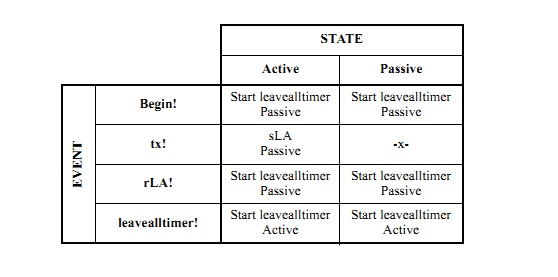


Abbildung 20: LeaveAll state machine[[3]](#footnote-3)

**Zustände:**

|  |  |
| --- | --- |
| Zustand | Bedeutung |
| Active | Automat ist bereit eine LeaveAll-Nachricht zu verschicken |
| Passive | Automat ist inaktiv |

**Events:**

|  |  |
| --- | --- |
| Event | Bedeutung |
| Begin! | Event zur Initialisierung des Automaten |
| tx! | Event, das anzeigt, dass nun eine Übertragungsmöglichkeit existiert |
| rLA! | Eine Externe LeaveAll-Nachricht wurde empfangen |
| leavealltimer! | Event verursacht, dass der Automat eine LeaveAll-Nachricht verschickt. |

**Actions:**

|  |  |
| --- | --- |
| Action | Bedeutung |
| sLA | Eine LeaveAll Nachricht wird ins Netz geschickt |
| Start leavealltimer | Nachdem der Timer abgelaufen ist, wird eine LeaveAll Nachricht verschickt |

### Zustandsautomat - Periodic Transmission state machine

**Zustandsautomat:**

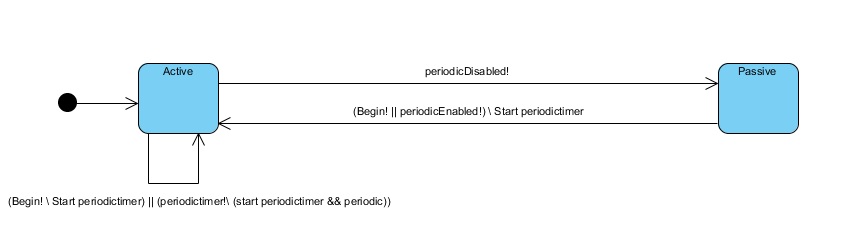


Abbildung 21: Zustandsautomat - PeriodicTransmission state machine

**Zustandsübergangstabelle:**

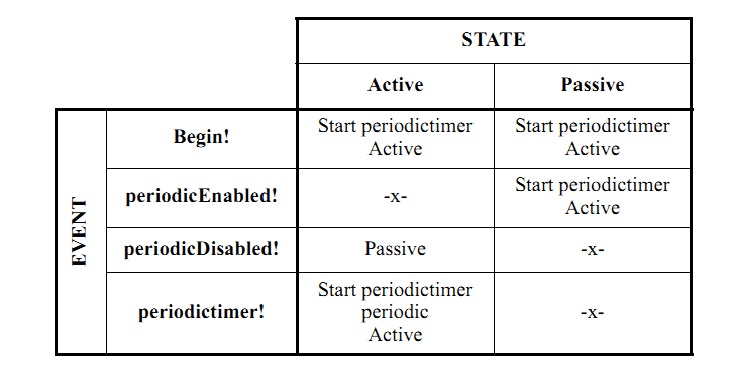


Abbildung 22: Periodic Transmission state machine[[4]](#footnote-4)

**Zustände:**

|  |  |
| --- | --- |
| Zustand | Bedeutung |
| Active | Automat ist aktiv |
| Passive | Automat ist inaktiv |

**Events:**

|  |  |
| --- | --- |
| Event | Bedeutung |
| Begin! | Der Automat wird initialisiert. |
| periodicEnabled! | Periodic Transmission state machine wurde aktiviert. |
| periodicDisabled! | Periodic Transmission state machine wurde deaktiviert. |
| periodictimer! | Automat schickt ein internes periodic-Event an die Applicant state machine. |

**Actions:**

|  |  |
| --- | --- |
| Action | Bedeutung |
| Start periodictimer | Nachdem der Timer abgelaufen ist, wird eine periodic Nachricht abgeschickt. |
| periodic | Sendet eine periodic Nachricht an die Applicant state machine, damit sie eine JoinIn Nachricht schickt. |

## GMRP-Paket

GMRP-Pakete sind gemäß der GARP-Struktur aufgebaut.

### GARP PDU Struktur

|  |  |
| --- | --- |
| 2 Byte |  |
| Protokoll ID | GARP Nachricht |

Protokoll ID (GARP): x01

### GARP Nachricht Struktur

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 Byte |  | | | |
| Attribut Typ | Attribut 1 | … | Attribut n | Endzeichen |

Endzeichen: x00

### Attribut Struktur

|  |  |  |
| --- | --- | --- |
| 1 Byte | 1 Byte | 1 Byte |
| Attribut Länge | Attribut Event | Attribut Wert |

Attribut Events:

0 Leave\_all

1 Join\_Empty

2 Join\_In

3 Leave\_Empty

4 Leave\_In

5 Empty

# Referenzen

[1] Verweis auf CRS „TINF11D\_CRS\_MultiCastor30\_Team\_4\_1v2“: Produkteinsatz

[2] Verweis auf SRS „TIT10AID\_SRS\_MCastor2v0\_Team\_4\_1v0.pdf“: Use Cases

[3] <http://de.wikipedia.org/wiki/Pcap> (22.10.2012): Pcap

# Produktfunktionen

# Anforderungen

# Produktcharakteristiken

## Systemumgebung

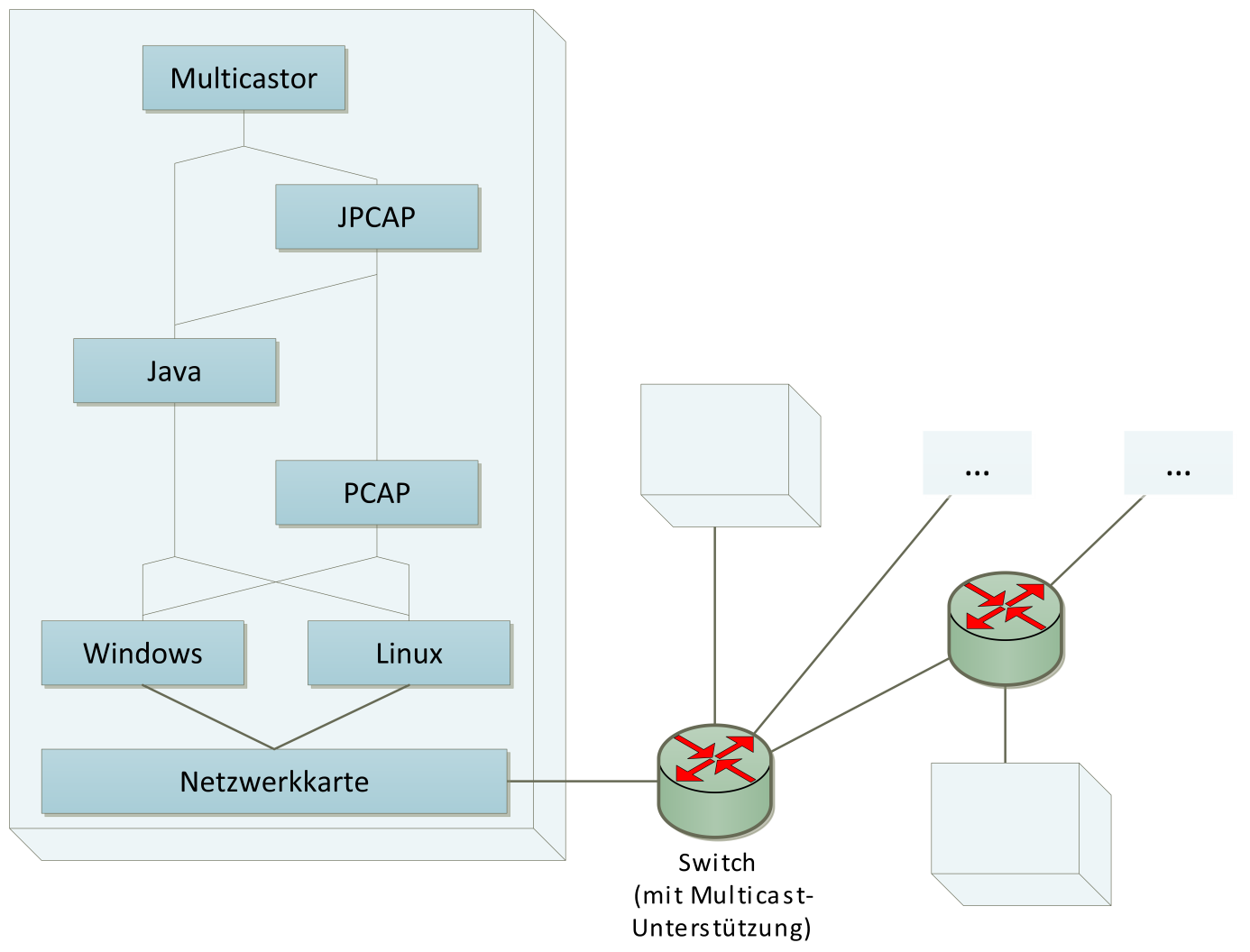


Abbildung 1: Übersicht über Systemumgebung

### Hardwareumgebung

Zum sinnvollen Testen von Multicast muss ein Netzwerk mit mehreren angeschlossenen Rechnern vorhanden sein. Auf Rechnerseite werden dabei alle halbwegs modernen Ethernetkarten unterstützt.

Um Multicast auf Layer 2 (MMRP und GMRP) zu nutzen muss allerdings ein Switch benutzt werden, der dies explizit unterstützt.

### Softwareumgebung

Multicastor baut auf Java auf. Unterstützt werden dabei Version 6 und neu in Multicastor 3.0 auch Version 7.

Um Multicast auf Layer 2 (MMRP und GMRP) zu nutzen muss zum einen PCAP und JPCAP, um PCAP in Java verwenden zu können, installiert sein und zum anderen Multicastor als Administrator gestartet werden.

Als Betriebssystem wird dabei entweder Windows 7, XP oder Linux ab Kernelversion 2.6.18 unterstützt.

## Nicht-funktionale Anforderungen

### LL10/ Fehlerhandling

Folgende größere Problemfelder sollen behoben werden:

|  |  |
| --- | --- |
| Titel | Beschreibung |
| Java 7 Kompatibilität | Die graphische Oberfläche ist bislang an mehreren Punkten zu Java 7 inkompatibel. |
| Messfehler | Die Messwerte sind fehlerhaft, der Messgraph zeigt selbst auf dem Loopback Device starke Schwankungen an. Außerdem bleibt die Empfangsrate bei abgeschaltetem Sender auf einem konstanten Wert über 0. Ein Grund ist der Java Garbage Collector. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### LL20/ Effiziente Nutzung

### LL30/ Benutzerfreundlichkeit

Um die Benutzerfreundlichkeit zu verbessern sind einige Änderungen und Verbesserungen an der GUI vorgesehen:

|  |  |
| --- | --- |
| Titel | Beschreibung |
| Tooltips | Es sind bereits einige Tooltips vorhanden, diese sollen aber in Anzahl und Inhalt noch deutlich ausgebaut werden. |
| Treeview | Eine Baumansicht soll die Messwerte verschiedener Sender einer Multicastgruppe in einer Überkategorie zusammenfassen. |
| Multithreading | Durch eine Ausführung in parallelen Threads wird die GUI stärker von der Berechnung getrennt, so dass erstere unter hoher Last besser ansprechbar bleibt. |
|  |  |
|  |  |
|  |  |
|  |  |

# Referenzen

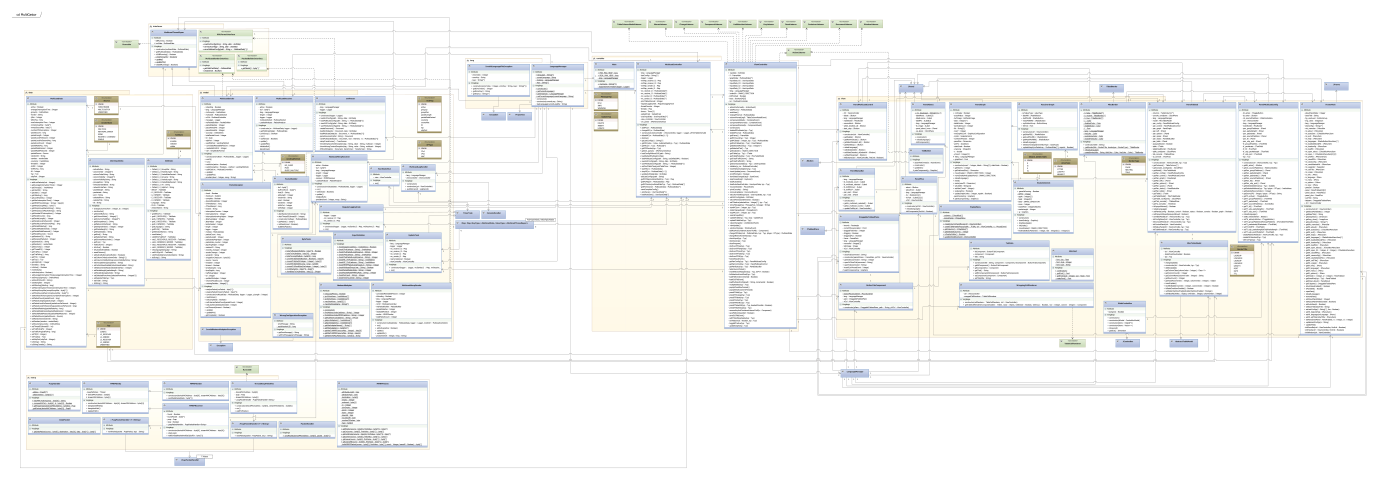


Abbildung 2:

1. Quelle: Analyse des MMRP-Protokolls nach IEEE 802.3ak einschließlich Erstellung einer zustandsorientierten Simulation des Protokolls auf Basis von OMNeT++ sowie weiterer Tools für Testzwecke von Johannes Jochen [↑](#footnote-ref-1)
2. Quelle: Analyse des MMRP-Protokolls nach IEEE 802.3ak einschließlich Erstellung einer zustandsorientierten Simulation des Protokolls auf Basis von OMNeT++ sowie weiterer Tools für Testzwecke von Johannes Jochen [↑](#footnote-ref-2)
3. Quelle: Analyse des MMRP-Protokolls nach IEEE 802.3ak einschließlich Erstellung einer zustandsorientierten Simulation des Protokolls auf Basis von OMNeT++ sowie weiterer Tools für Testzwecke von Johannes Jochen [↑](#footnote-ref-3)
4. Quelle: http://standards.ieee.org/getieee802/download/802.1ak-2007.pdf Seite 58 [↑](#footnote-ref-4)