Konzepte der internetbasierten Gerätesteuerung

Von der Fakultät für Mathematik und Informatik

der Technischen Universität Bergakademie Freiberg

genehmigte

D i s s e r t a t i o n

zur

Erlangung des akademischen Grades

Doktor-Ingenieur

Dr.-Ing.

vorgelegt

von Dipl.-Ing. B.Sc. Gunnar Hoffmann

geboren am 20.07.1979 in Aachen

Gutachter: Prof. Dr. Konrad Froitzheim, Freiberg
Prof. Dr. Albrecht Schmidt, Stuttgart

Tag der Verleihung: 05. Dezember 2011

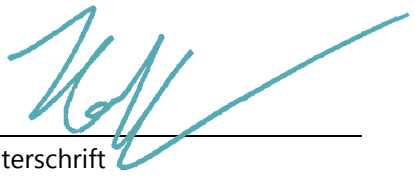
Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Bei der Auswahl und Auswertung des Materials sowie bei der Herstellung des Manuskripts habe ich keine Unterstützungsleistungen Dritter, wie Promotionsberater, in Anspruch genommen. Weitere Personen haben von mir keine geldwerten Leistungen für Arbeiten erhalten, die nicht als solche kenntlich gemacht worden sind.

Die Arbeit wurde bisher weder im Inland noch im Ausland in gleicher oder ähnlicher Form einer anderen Prüfungsbehörde vorgelegt.

Wuppertal, 14.04.2011
Ort, Datum



Unterschrift

Pour Ellen

Verbringe nicht die Zeit mit der Suche nach einem Hindernis.
Vielleicht ist keines da.

Franz Kafka (1883–1924)

Danksagung

Die vorliegende Arbeit wurde im Rahmen einer externen Promotion an der Fakultät für Mathematik und Informatik der Technischen Universität Bergakademie Freiberg angefertigt.

Mein Dank gilt an erster Stelle Herrn Professor Dr. Froitzheim, der das meinen Neigungen entsprechende Promotionsthema initiierte. Trotz seiner zahlreichen Verpflichtungen folgte er den Fortschritten der Arbeit mit viel Engagement. Seine Anregungen und seine Diskussionsbereitschaft waren eine große Bereicherung. Ebenso danke ich Herrn Professor Dr. Schmidt für seine zugleich herausfordernden und hilfreichen Kommentare.

Wuppertal, im April 2011 Gunnar Hoffmann

Inhalt

1 Einleitung 1
1.1 Zielstellung der Arbeit 1
1.2 Aufbau der Arbeit 4
2 Verwandte Arbeiten und Architekturfundament 7
2.1 Lokale Netze 9
2.1.1 Ethernet 9
2.1.2 WLAN und WPAN 10
2.1.3 Feldbussysteme 15
2.1.4 KNX/EIB 19
2.2 Verteilte Ad-Hoc-Systeme 20
2.2.1 Jini 21
2.2.2 Zeroconf 26
2.2.3 UPnP 36
2.2.4 IEEE 1451 43
2.2.5 Weitere Ansätze 55
2.3 Geräteidentifikation 56
2.3.1 OCR 57
2.3.2 RFID 58
2.3.3 Barcode 59
2.4 Audio- und Video-Streaming 60
2.4.1 Progressive Streaming 60
2.4.2 Realtime Streaming 61
2.4.3 Streaming-Server und -Clients 64
3 Petrinetze 69
3.1 Grundlegende Eigenschaften 71
3.2 Dynamisches Verhalten 75
3.3 Invariantenkalkül 77
3.4 Beispiel 81
4 Konzepte 85
4.1 Analyse 86
4.2 Architektur 89
4.2.1 Interface von DevControl-Geräten 92
4.2.2 Interface von DevControl-Gateways 94
4.2.3 Interface von Mediastreams 95
4.2.4 Interface von Geräten für die Autoidentifikation 96

4.3 DevXML 97
4.3.1 Datenmodell 97
4.3.2 Sprachelemente 98
4.3.3 Navigation 117
4.3.4 Modellzuverlässigkeit 126
5 Referenzimplementierung 131
5.1 Browser-Plugin 131
5.2 Gateway 132
5.3 Mediastreams 136
5.4 Autoidentifikation 138
6 Experimentelle Validierung anhand ausgewählter DevControl-Geräte 141
6.1 Switch 144
6.2 Heating 146
6.3 Multimeter 149
6.4 Phone 151
6.5 Terminal 153
6.6 Jukebox 155
6.7 Alarm 160
6.8 Virtual Devices 161
6.8.1 Haptic Phone 162
6.8.2 Automated Switch 163
6.9 Abschließende Bewertung 164
7 Zusammenfassung und Ausblick 165
Quellenverzeichnis 169
Anhang 179
A0 TEDS am Beispiel eines Drucksensors 179
A1 DevXML Schemadefinition 183
A2 DevXML-Code Switch 186
A3 DevXML-Code Heating 187
A4 DevXML-Code Multimeter 189
A5 DevXML-Code Phone 190
A6 DevXML-Code Jukebox 191
A7 DevXML-Code Alarm 194
A8 DevXML-Code Haptic Phone 195
A9 DevXML-Code Automated Switch 196

Kapitel 1

Einleitung

Die Durchdringung unterschiedlicher Anwendungsdomänen mit über das Internet steuerbaren Geräten schreitet fort. Die gegenwärtigen Systeme sind durch ein hohes Maß an Heterogenität gekennzeichnet. Die Herausbildung proprietärer Lösungen führt zu funktionalen Einschränkungen und zu einem inhomogenen Erscheinungsbild gegenüber Nutzern. Dies steht dem Grundgedanken einer anwenderzentrierten Arbeitsumgebung entgegen.

1.1 Zielstellung der Arbeit

Bei heute verfügbaren Geräten fehlt als verbindendes Element offensichtlich ein einheitliches Architekturkonzept, bei dem die Autokonfiguration, die Gerätezuordenbarkeit vor Ort, die Geräteüberwachbarkeit, die Inter-Gerätekommunikation und die Automatisierbarkeit von Abläufen ausgewogen Berücksichtigung finden.

Es ist Ziel der vorliegenden Arbeit, diese Lücke zu schließen. Im Folgenden wird die Diskrepanz zwischen Anspruch und Wirklichkeit in Bezug auf obige Punkte skizziert. Abbildung 1 vermittelt einen Eindruck von der avisierten als Device Control (DevControl) bezeichneten Architektur. Geräte wie Lampen, Webcams und Jukeboxes kommunizieren über ein einheitliches Protokoll und geben ihre Funktionalität über ein Gateway bekannt. Der Gerätebegriff wird im weiteren Sinne verwendet und umfasst gleichermaßen Anwendungen. Für die Steuerung sind gängige Webbrowser vorgesehen.

DevControl ist gleichermaßen für das industrielle, gewerbliche und private Umfeld bestimmt. Hiermit werden die Bereiche Smart Metering und Smart Home, die in der aktuellen energiepolitischen Diskussion eine herausgehobene Stellung einnehmen, vollständig überdeckt. Die Motivation für den Einsatz von Steuerungslösungen ist vielschichtig. Sie wird von DevControl nicht vorbestimmt. Ob im konkreten Einzelfall beispielsweise Komfortzugewinn oder Energieeinsparungen überwiegen, ist aus technologischer Sicht unerheblich. Auf spezielle Teilbereiche – wie z. B. E-Energy – wird aus diesem Grund nicht explizit eingegangen.

Der unmittelbaren Nutzung von Geräten stehen in der Phase der erstmaligen Inbetriebnahme häufig aufwändige Netzwerkeinstellungen entgegen. Diese sind in der Regel herstellerspezifisch. Es wird untersucht, welche der bereits existierenden Infrastrukturtechnologien sich für die Autokonfiguration eignen. Der Fokus ist auf die Technologien Jini, Zeroconf, UPnP und die aus dem Bereich der Mess-, Steuer- und Regeltechnik stammende Standardfamilie IEEE 1451 gerichtet.

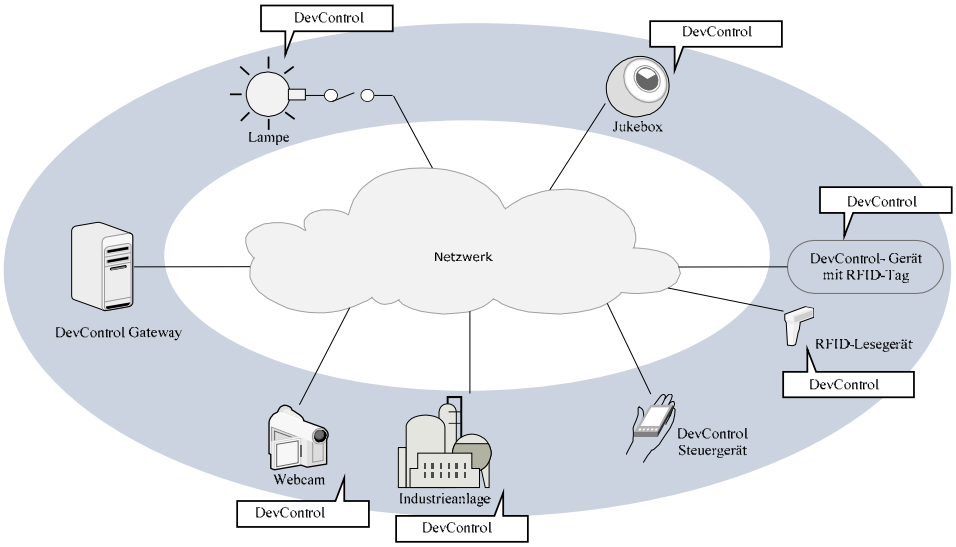


Abbildung 1 Beispielhaftes Anwendungsszenario der generischen Gerätesteuerung. Die Kommunikation erfolgt auf Basis von DevControl. Als zentrale Instanz dient ein Gateway, das wesentliche Geräteinformationen aufbereitet und Steuergeräten zur Verfügung stellt.

Für die Gerätezuordenbarkeit vor Ort wird ein Ansatz zur nahtlosen Einbindung von Identifikationstechnologien wie OCR, RFID und Barcode erarbeitet. Dieser basiert auf der Speicherung von Gerätekennungen auf Identifikationsobjekten. Durch geeignete Integration von Erfassungsgeräten lässt sich das zu steuernde Gerät dann lokal identifizieren. Somit besteht eine benutzerfreundliche Alternative zur Listenauswahl.

Die Steuerung von Geräten aus der Ferne ist häufig mit deren Überwachung verbunden, um so mögliche Abweichungen vom Sollverhalten feststellen zu können. Hierzu eignet sich Streaming. Aufgrund der herausgehobenen Bedeutung – insbesondere im Umfeld sicherheitskritischer Anwendungen – ist die direkte Integration von Audio- und Videostreams in DevControl beabsichtigt. Es wird darauf hingewirkt, ein breites Band von Steuergeräten für die Medienwiedergabe einsetzen zu können.

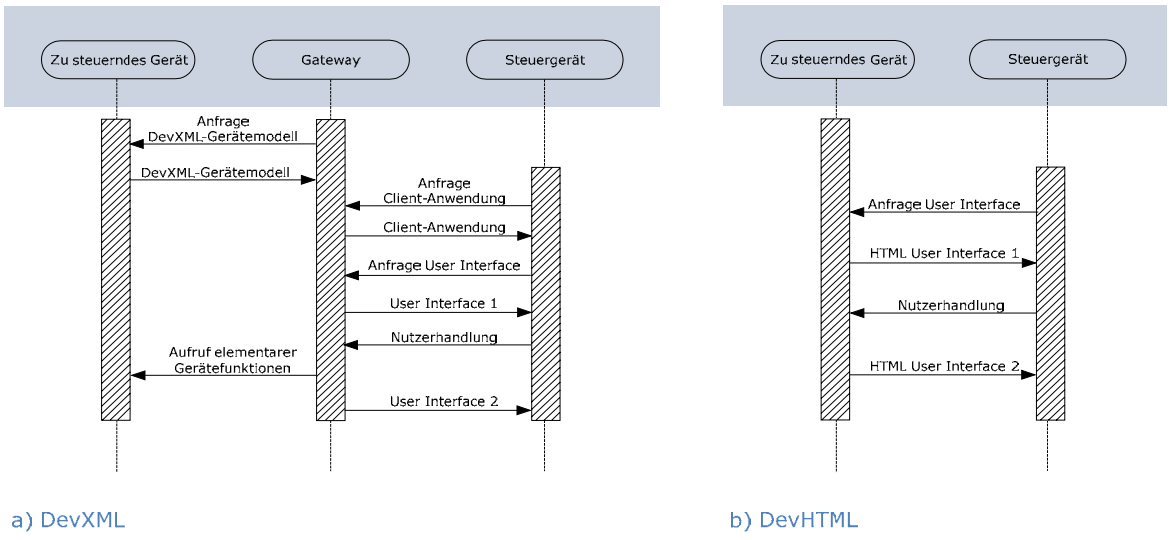
Über das Internet steuerbare Geräte sind heute bereits verfügbar, jedoch werden von den Herstellern hinsichtlich der zur Anwendung kommenden Protokolle abweichende – zumeist proprietäre – Ansätze verfolgt. Hiermit verbunden ist ein nur schwach ausgeprägter Interaktionsgrad. Die Bildung komplexer Gesamtsysteme, die sich aus einer Vielzahl von Einzelgeräten verschiedener Hersteller zusammensetzen, bedingt aufwändige Anpassungen und ist daher auf Individuallösungen beschränkt. Mit dem Konzept der „Virtual Devices“ ermöglicht DevControl hingegen die Einbettung von Teilfunktionalitäten mehrerer Geräte in einen neuen Kontext. Die Modellierung eines geräteübergreifenden „Virtual Device“ kann den gesamten Funktionsumfang der referenzierten DevControl-konformen Geräte einbeziehen. Zu nennen sei beispielsweise ein „Virtual Device“, das sich aus einer Photodiode und einem Lichtschalter zusammensetzt und die Funktion der Tag-Nacht-abhängigen Außenlichtsteuerung realisiert.

Viele Geräte bieten ausschließlich Funktionen, die unmittelbar durch den Benutzer auszuführen sind. Es sind manuelle Handlungen zu tätigen wie z. B. das Ein- und Ausschalten eines Lichts. In vielen Anwendungsbereichen ist es hingegen sinnvoll, darüber hinaus bestimmte Abhängigkeiten in Form von Regeln zu definieren, um so Abläufe zu automatisieren. DevControl sieht derartige Regeldefinitionen vor. Auf diese Weise ist die zeit- und zustandsbasierte Ausübung von Funktionen möglich.

Zu erstgenannter Gruppe zählt beispielsweise ein Licht, das zu im Vorfeld definierten Zeiten automatisch ein- bzw. ausgeschaltet wird. Der letztgenannten Gruppe lässt sich das bereits im Zusammenhang mit dem Interaktionsgrad genannte Beispiel der Tag-Nacht-abhängigen Außenlichtsteuerung zuordnen.

Die zu DevControl gehörige Modellierungssprache Device XML (DevXML) schafft die Voraussetzung für die Inter-Gerätekommunikation und Automatisierbarkeit von Abläufen. Erreicht wird dies durch die Entkopplung von Gerätefunktionalität und User Interface. Petrinetze bilden hierbei die theoretische Grundlage für die Modellierung von Abhängigkeiten zwischen Gerätefunktionen und -zuständen. Elemente zur Beschreibung der Nutzerschnittstelle sind auf einer technologieunabhängigen Abstraktionsebene angesiedelt. Als Zwischeninstanz erfordert das Konzept den Einsatz eines Gateways. Im zeitlichen Ablauf stellt sich die Kommunikation hierbei wie in Abbildung 2 a) dar.

Auf Anfrage wird dem Gateway das Modell des zu steuernden Gerätes übermittelt. Das Gateway wiederum stellt dem Steuergerät zur Laufzeit eine Client-Anwendung bereit, mittels derer die in der Folge übermittelten User-Interface-Informationen darstellbar sind. Im Übrigen bestimmt das Gateway die infolge von Nutzerhandlungen aufzurufenden Gerätefunktionen und liefert resultierende User-Interface-Informationen an das Steuergerät zurück. Ein weiterer wesentlicher Vorzug von DevXML ist die Technologieoffenheit. Das User Interface kann unter Berücksichtigung des Stands der Technik den jeweiligen Erfordernissen angepasst werden, ohne hierbei Änderungen am Gerät selbst vorzunehmen. Wahlweise lassen sich beispielsweise User Interfaces mit VoiceXML, XUL oder Silverlight erzeugen. Bei der prototypischen Realisierung im Rahmen der Arbeit wird auf Silverlight zurückgegriffen. Zu den Merkmalen von DevXML zählt die einfache Erweiterbarkeit der Sprachdefinition.



a) DevXML b) DevHTML

Abbildung 2 Die Steuerung von DevControl-Geräten kann mittels DevXML und DevHTML vorgenommen werden. Bei DevXML sind Gerätefunktionalität und User Interface strikt voneinander getrennt mit dem Ziel, die Inter-Gerätekommunikation sowie die Automatisierbarkeit von Abläufen zu ermöglichen. Fall a) stellt die wesentlichen Schritte der mit DevXML verbundenen Kommunikation dar. Unter Rückgriff auf das formale Modell des zu steuernden Gerätes werden vom Gateway die für das Steuergerät bestimmten Informationen zur User-Interface-Generierung bereitgestellt. Die hierbei zum Einsatz kommende Technologie ist frei wählbar, es eignen sich beispielsweise VoiceXML, XUL und Silverlight. Fall b) zeigt die DevHTML-basierte Kommunikation. Für die Nutzung der grundlegenden Funktionalität stellt das zu steuernde Gerät HTML User Interfaces bereit. Weder die Inter-Gerätekommunikation noch die Automatisierbarkeit von Abläufen werden von DevHTML unterstützt.

Durch die freie Wählbarkeit der Technologie für das User Interface variieren bei DevXML die Anforderungen an Steuergeräte. Mit der Absicht, die Nutzung von Grundfunktionen unter im Vorfeld definierten Randbedingungen zu gewährleisten, ist die alternative Steuerung mittels Device HTML (DevHTML) vorgesehen. Abbildung 2 b) gibt einen exemplarischen Kommunikationsablauf wieder. Hierbei fordert das Steuergerät ein HTML-basiertes User Interface direkt vom zu steuernden Gerät an, dessen grundlegende Operationen auf diese Weise zugänglich gemacht werden. Nach außen hin bleibt verborgen, welche Funktionsaufrufe mit Nutzerhandlungen verbunden sind. Mittels DevHTML ist daher weder die Inter-Gerätekommunikation noch die Automatisierbarkeit von Abläufen möglich. Als Ergebnis einer Interaktion liefert das zu steuernde Gerät unmittelbar ein aktualisiertes User Interface.

DevControl-Geräte stellen die mit ihnen verbundenen DevXML- und DevHTML-Beschreibungen über definierte Schnittstellen bereit. Neben der Ausarbeitung des integralen DevControl-Architekturkonzepts ist dessen Implementierung Bestandteil der Arbeit. Überdies werden ausgewählte DevControl-Geräte realisiert. Unterschiedliche Hardwareplattformen finden Berücksichtigung.

1.2 Aufbau der Arbeit

Ausgehend vom vorgenannten Architekturzielkonzept folgen im Kapitel 2 detaillierte Ausführungen zu Hilfstechnologien, die potentiell das Architekturfundament bilden. Der infrastrukturbezogene Themenkomplex ist weiter untergliedert. Zunächst werden in Kapitel 2.1 unterschiedliche Typen lokaler Netze hinsichtlich ihrer Eignung im Kontext einer internetbasierten Gerätesteuerung mit dem Anspruch auf Generik evaluiert. In Kapitel 2.2 „Verteilte Ad hoc Systeme“ sind Technologien, die selbstkonstituierende Systemverbünde unterstützen, Gegenstand der Untersuchungen. In Kapitel 2.3 wird beschrieben, wie mit bereits verfügbaren Produktlösungen die Geräteidentifikation vor Ort realisiert werden kann. Die thematische Befassung mit der Einbindung von Audio- und Video-Streaming in das ganzheitliche Konzept erfolgt in Kapitel 2.4.

In Kapitel 3 wird die Systemmodellierung mittels Petrinetzen behandelt. Zunächst werden deren grundlegende Eigenschaften in Kapitel 3.1 diskutiert. Vertiefende Ausführungen zum dynamischen Verhalten von Petrinetzen in Kapitel 3.2 schaffen die Basis für das Verständnis der Erreichbarkeitsanalyse. Diese ist für die geführte Navigation im Rahmen von DevXML von maßgeblicher Bedeutung. Invariante Eigenschaften von Petrinetzen lassen sich z. B. für die Überprüfung von Markierungen auf Gültigkeit nutzen. Aus diesem Grund wird auf das zugrunde liegende Kalkül in Kapitel 3.3 näher eingegangen. Beispielrechungen in Kapitel 3.4 bilden den Abschluss der netztheoretischen Betrachtungen.

Es folgt die Vorstellung des Lösungskonzeptes in Kapitel 4. Mit der einführenden Analyse in Kapitel 4.1 wird die Auswahl der zum Einsatz kommenden Hilfstechnologien erläutert. Die Beschreibungen zur Architektur DevControl in Kapitel 4.2 gliedern sich weiter in die Unterkapitel 4.2.1–4.2.4. Thematische Schwerpunkte sind Schnittstellen von DevControl-Geräten, DevControl-Gateways, Mediastreams und Geräten für die Autoidentifikation. Im Vordergrund des Kapitels 4.3 steht die entwickelte XML-Sprache DevXML als tragendem Element. Die zugehörigen Unterkapitel 4.3.1–4.3.4 umfassen Details zum Datenmodell, zu Sprachelementen, zur Navigation sowie zur Modellzuverlässigkeit.

Gegenstand des Kapitels 5 ist die entwickelte Referenzimplementierung, mit der das DevControl-Architekturkonzept prototypisch umgesetzt wird. Sie stellt die Grundvoraussetzung für die Nutzung von DevControl-konformen Geräten dar. Die Ausführungen zu den Bestandteilen Browser-Plugin, Gateway, Mediastreams und Autoidentifikation sind in die Unterkapitel 5.1–5.4 gegliedert. Kapitel 6 befasst sich mit der Validierung des integralen Frameworks anhand ausgewählter DevControl-Geräte, denen die Unterkapitel 6.1–6.8 gewidmet sind. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

Kapitel 2

Verwandte Arbeiten und Architekturfundament

Einleitend wurde auf die wesentlichen Charakteristika des avisierten Architekturkonzepts eingegangen. Gegenstand des Kapitels 2 sind Technologien mit Infrastrukturbezug. Im Mittelpunkt der Betrachtung steht deren mögliche Eignung, einen Aufsetzpunkt für das zu entwickelnde generische Architekturkonzept DevControl darzustellen. Mit dem Ziel, hierfür eine objektive Bewertungsgrundlage zu schaffen, werden relevante Technologiemerkmale herausgearbeitet. Das Themenfeld umfasst „Lokale Netze“, „Verteilte Ad hoc Systeme“, „Geräteidentifikation“ sowie „Audio- und Video-Streaming“. Siehe hierzu Abbildung 3. Sie stellt alle Technologien, die in den Kapiteln 2.1–2.4 vertiefend behandelt werden, in der Übersicht dar.

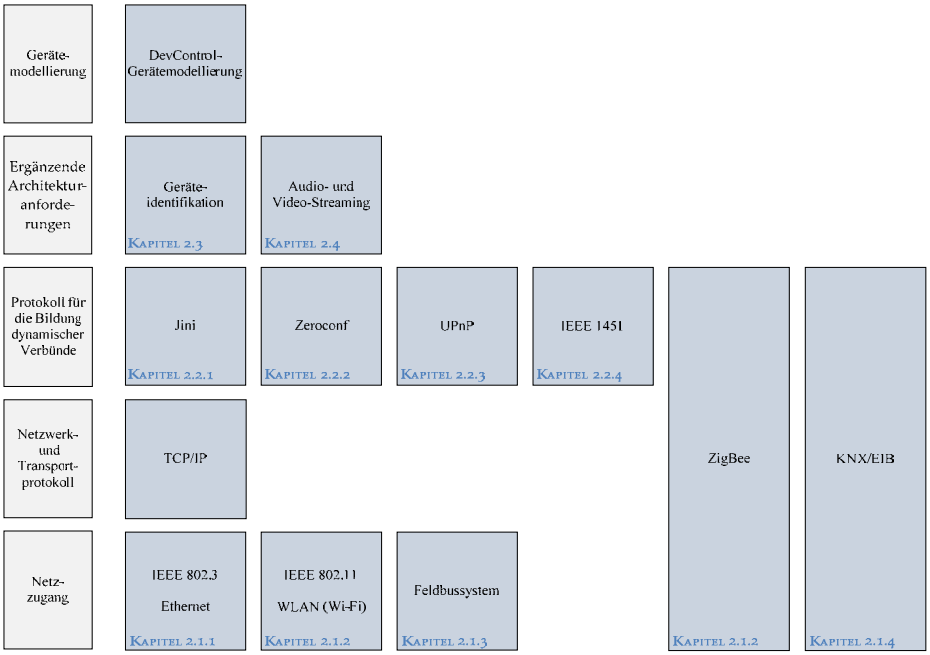


Abbildung 3 Das potentielle Fundament der Zielarchitektur DevControl erstreckt sich vom Netzzugang über die Autokonfiguration bis hin zur Geräteidentifikation vor Ort und Streaming. Diesbezüglich werden die hier aufgeführten – bereits existenten – Technologien auf ihre Eignung untersucht.

Der Grad der Vernetzung elektronischer Geräte – auch solcher des alltäglichen Lebens – hat in den letzten Jahren signifikant zugenommen. Damit ist die Evolution hin zum „Internet of Things“ eingeleitet. Shirazi et al. zeigen, wie unter Nutzung von Webprotokollen der Zugriff auf Sensordaten bewerkstelligt werden kann[1]. In der Hauptsache wird das Problem der Informationsbeschaffung und -bereitstellung in WANs gelöst.

Komplementär hierzu ist die generische Steuerung von Geräten zu sehen, z. B. von Lichtschaltern, Projektoren und Rollläden[2,3]. Aktuelle Arbeiten zum Themenkomplex sind überwiegend auf einzelne Teilaspekte beschränkt. Somit stehen beispielsweise häufig die Ausbildung von Ad-Hoc-Infrastrukturen[4,5] und die Generierung von User Interfaces[6] unverbunden nebeneinander. Im Rahmen der eigenen Ausführungen werden zentrale Ideen aufgegriffen und zusammengeführt mit dem Bestreben, eine gesamthafte Architektur mit Anspruch auf Generik zu realisieren. Hinsichtlich der Auswahl eines verteilten Ad-Hoc-Systems finden Untersuchungen zu den unterschiedlichsten Technologien mit deren Vorteilen, Nachteilen und Entwicklungspotentialen Berücksichtigung[7,8,9,10].

Vielmals steht die Forderung nach gesicherter Kommunikation. Auf die diesbezüglich verwendbaren Verfahren wird nicht im Detail eingegangen, da die Problematik in zahlreichen Publikationen erschöpfend behandelt wird. Dies schließt Mechanismen zur Authentifizierung und zur Verhinderung der unbefugten Daten-Einsichtnahme sowie -Manipulation mit ein[11,12].

Wie oben erwähnt ist die Geräteerkennung vor Ort in das DevControl-Konzept eingepflochten. Ailisto et al. diskutieren diese Form der Zuordenbarkeit, wobei sie eine Untergliederung in die Paradigmen ScanMe, PointMe und TouchMe vornehmen[13]. Ein konkreter Architekturvorschlag wird allerdings nicht unterbreitet. Damit grenzen sich die eigenen Ausarbeitungen maßgeblich ab.

Auch die nahtlose Integration von Audio- und Video-Streaming in Ad-Hoc-Umgebungen ist Gegenstand aktueller Forschung. Die übergeordneten Zielstellungen sind vielschichtig. Hierzu zählt beispielsweise die Realisierung offener Lösungen für die IP-basierte Video- und Sprachtelefonie[14]. Vorliegend kommt der Darstellbarkeit von Streaminginhalten auf einer Vielzahl mobiler Endgerätearten eine hohe Bedeutung zu. Aus diesem Grund umfasst die Diskussion mehrere praxisrelevante Ansätze, hierunter Streaming über HTTP[15].

---

[1] Shirazi/Winkler/Schmidt: SENSE-SATION – An Extensible Platform for Integration of Phones into the Web.
[2] Wolf/Froitzheim/Weber: Interactive Video and Remote Control via the World Wide Web.
[3] Shirehjini: A generic UPnP architecture for ambient intelligence meeting rooms.
[4] Tayal/Patnaik: An address assignment for the automatic configuration of mobile ad hoc networks.
[5] Fors et al.: Ad-hoc Composition of Pervasive Services in the PalCom Architecture.
[6] Nichols/Myers: Creating a lightweight user interface description language.
[7] Wei et al.: Use of the ‚smart transducer' concept and IEEE 1451 standards.
[8] Kochan et al.: Approach to improving network capable application processor based on IEEE 1451 standard.
[9] Harihar/Kurkovsky: Using Jini to enable pervasive computing environments.
[10] Watanabe et al.: A scheme of service discovery and control on ubiquitous devices.
[11] Adida: SessionLock: Securing Web Sessions against Eavesdropping.
[12] Sales et al.: Enabling user authentication and authorization to support context-aware UPnP applications.
[13] Ailisto et al.: Bridging the physical and virtual worlds by local connectivity-based physical selection.
[14] Vilei/Convertino/Crudo: A new UPnP architecture for distributed video voice over IP.
[15] Stockhammer: Dynamic adaptive streaming over HTTP.

2.1 Lokale Netze

Lokale Netze sind Kommunikationssysteme mit begrenzter Ausdehnung. Sie sind z. B. auf Gebäudekomplexe beschränkt. Im Kontext dieser Arbeit bildet ein lokales Netz die Basis für die Inter-Kommunikation der hieran angebundenen Geräte. Der Forderung nach ortsungebundener Gerätesteuerung mit Hilfe eines universell einsetzbaren Clients kann durch Anbindung des lokalen Netzes an das Internet nachgekommen werden. Für die technische Ausgestaltung des lokalen Netzes kommen mehrere Realisierungsoptionen in Betracht. Im Folgenden werden die Standardtechnologien Ethernet, WLAN, Zigbee, Feldbussystem und KNX/EIB näher untersucht. Auf dieser Grundlage ist dann die objektivierte Auswahl einer Infrastrukturtechnologie für DevControl möglich.

2.1.1 Ethernet

Ethernet ist die Bezeichnung für Netze mit dem dezentralen Medienzugriffsverfahren CSMA (Carrier Sense Multiple Access) mit Kollisionserkennung CD (Collision Detection). Das Verfahrensprinzip besteht darin, dass eine sendewillige Station den Kanal auf Signale anderer Stationen abhört. Wird als Ergebnis die Nichtbelegung des Kanals festgestellt, wird der Sendevorgang eingeleitet. Bei CSMA wird zwischen drei Grundvarianten differenziert:

- Nonpersistent CSMA: Das Medium wird in zeitdiskreten Abständen abgehört (Schritt 1). Ist das Medium nicht belegt, so wird der Sendevorgang sofort eingeleitet. Bei Belegung wird mit Schritt 1 fortgefahren.
- 1-Persistent CSMA: Das Medium wird kontinuierlich abgehört. Sobald es frei ist, erfolgt das Senden (p-Persistent mit $p = 1$).
- p-Persistent CSMA: Das Medium wird kontinuierlich abgehört (Schritt 1). Sobald es frei ist, erfolgt der Sendevorgang mit der Wahrscheinlichkeit p, andernfalls wird mit der Wahrscheinlichkeit p-1 für eine definierte Zeitdauer gewartet. Bei Eintritt des zuletzt genannten Falls wird im Anschluss mit Schritt 1 fortgefahren.

Es kann vorkommen, dass mehrere Stationen gleichzeitig den Versuch unternehmen, Daten zu übermitteln. Die Folge ist dann eine Signalüberlagerung, die durch die sendenden Stationen detektiert wird (Collision Detection). In diesem Fall erfolgt ein erneutes Senden durch jede einzelne der involvierten Stationen verzögert um eine stochastisch ermittelte Wartedauer (Backoff) mit dem Ziel, mehrmalig nacheinander auftretende Kollisionen zu vermeiden. Im Ethernet nach IEEE 802.3, dem das Verfahren 1-Persistent CSMA zugrunde liegt, wird ein exponentieller Backoff verwendet, dessen Mittelwert mit jeder Kollision verdoppelt wird. Ein Abbruch erfolgt nach 16 erfolglosen Zugriffsversuchen. Die Ethernet-Familie umfasst zahlreiche Varianten. Bei der Evolution ist die Tendenz zu immer höheren Datenraten klar erkennbar.

Topologische Ausprägungen des Ethernet beschränkten sich ursprünglich auf den Bus. Bei der ältesten Variante 10BASE5 ist das Buskabel beidseitig mit einem 50 Ω-Widerstand abgeschlossen. Mit dem Einsatz von Switches sowie getrennten Adernpaaren für das Senden und Empfangen konnten schließlich Sterntopologien ausgeführt und somit eine Abkehr vom shared medium hin zum dedicated medium bei gleichzeitiger Vollduplex-Eignung vollzogen werden. Zur Kategorie des Vollduplex-Ethernet zählen 100BASE-FX, 100BASE-TX und 100BASE-T2. Obwohl hierbei ausführungsbedingt keine Kollisionen mehr auftreten können, wird das CSMA/CD-Verfahren weiterhin beibehalten. Abbildung 4 zeigt den TCP/IP-Stack auf Basis von Ethernet.

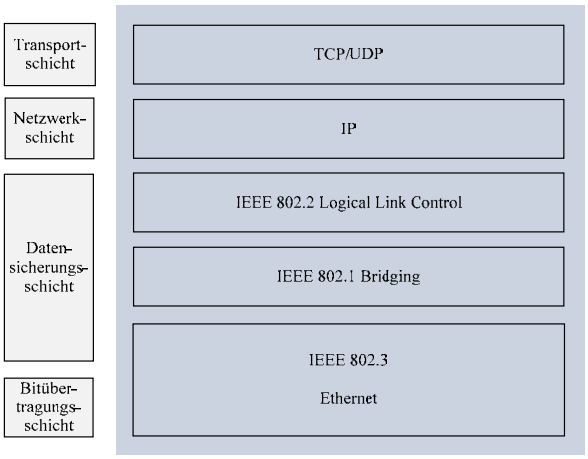


Abbildung 4 TCP/IP-Stack auf Grundlage von Ethernet. Die Spezifikationen der Datensicherungs- und Bitübertragungsschicht umfassen IEEE 802.2, IEEE 802.1 sowie IEEE 802.3.

Die Ethernet-Spezifikation erstreckt sich über die gesamte Bitübertragungsschicht und die untere Subschicht der Sicherungsschicht (MAC-Subschicht), die über Bridging mit der oberen Subschicht der Sicherungsschicht (Logical Link Control-Subschicht) verbunden ist. Die Anbindung eines Ethernet-basierten Netzwerkes an das Internet kann erfolgen, indem auf den OSI-Schichten 3 (Netzwerkschicht/Vermittlungsschicht) und 4 (Transportschicht) das TCP/IP-Protokoll zum Einsatz kommt[16].

2.1.2 WLAN und WPAN

Bei Wireless Local Area Networks (WLAN) und Wireless Personal Area Networks (WPAN) erfolgt die Übertragung von Daten vorzugsweise im lizenzierungsfreien ISM (Industrial, Scientific and Medical) 2,4 GHz-Frequenzband. Es dominieren WLAN gemäß IEEE 802.11, Bluetooth gemäß IEEE 802.15.1 und die auf IEEE 802.15.4 basierende Technologie ZigBee.

IEEE 802.15.1 definiert mehrere Geräteklassen, die sich in ihren technischen Parametern stark unterscheiden. Daher wird im Folgenden kurz auf die wesentlichen Unterschiede eingegangen. Die verfügbaren Leistungsklassen der Funktechnik Bluetooth lassen überbrückbare Distanzen bis 100 m zu. Es wird in Klasse 3 mit $P_{max} = 1$ mW, Klasse 2 mit $P_{max} = 2{,}5$ mW und Klasse 1 mit $P_{max} = 100$ mW unterschieden[17]. Die Datenrate eines Kanals beträgt bei Bluetooth 2.1 maximal 780 kbit/s.

Die Allokation von Zeitschlitzen durch die Kommunikationsteilnehmer erfolgt dynamisch in Abhängigkeit des Datenaufkommens. Im bidirektionalen Modus der asymmetrischen Datenübertragung steht einem Endgerät eine maximale Übertragungsgeschwindigkeit von 723 kbit/s zur Verfügung. Die Datenübermittlung in die Gegenrichtung ist aufseiten des Kommunikationspartners in diesem Fall auf 57 kbit/s begrenzt. Mit der im Jahre 2004 erschienenen Erweiterung Enhanced Data Rate (EDR) vervielfacht sich die von einem Endgerät erzielbare Übertragungsgeschwindigkeit auf 2178 kbit/s. Der korrespondierende Wert des Kommunikationspartners beträgt dann 177 kbit/s[18].

[16] Tanenbaum: Computernetzwerke, S. 60.
[17] Gessler/Krause: Wireless-Netzwerke für den Nahbereich, S. 191.
[18] Sauter: Grundkurs Mobile Kommunikationssysteme, S. 345 ff.

Diese sind mit ihren Frequenzbändern, Reichweiten und Übertragungsraten in Tabelle 1 aufgeführt[19,20,21,22]. Durch die Wireless Ethernet Compatibility Alliance (WECA) zertifizierte IEEE 802.11-konforme Geräte tragen das Gütesiegel WiFi (Wireless Fidelity).

Tabelle 1 WLAN- und WPAN-Varianten.

Standard	Frequenzband in GHz	Reichweite innerhalb von Gebäuden in m	Bruttoübertragungs-rate in Mbit/s
	WLAN		
IEEE 802.11b	2,4	< 35	11
IEEE 802.11a	5	< 35	54
IEEE 802.11g	2,4	< 35	22 und 54
IEEE 802.11n	2,4; 5	< 70	600
	WPAN		
IEEE 802.15.1 (Bluetooth 2.1 mit EDR)	2,4	< 100	3
ZigBee Alliance (basiert auf IEEE 802.15.4)	2,4 (weltweit); 0,868 (Europa); 0,915 (USA)	< 40	0,25

Hinsichtlich der Topologie in einem WLAN/WPAN wird unterschieden in Infrastruktur- und Ad-Hoc-Modus. Im zentralistischen Infrastrukturmodus wird die Kommunikation über einen festen Zugangspunkt geleitet. Im Ad-Hoc-Modus bildet sich ein Peer-to-Peer-Netzwerk aus, bei dem die Teilnehmer direkt untereinander kommunizieren. Im Gegensatz zu den beschriebenen drahtgebundenen Technologien sind WLAN/WPAN-Stationen in der Regel so konzipiert, dass sie nur eine Funkeinheit beinhalten und somit nicht zeitgleich senden und empfangen können.

Der bereits vorgestellte CSMA/CD-Mechanismus, bei dem während des Kommunikationsprozesses das Medium durch den Sender abgehört wird, kann daher nicht verwendet werden. An die Stelle der Kollisionserkennung rückt bei WLAN/WPAN die Kollisionsvermeidung. Hierbei kommen einfaches CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance), CSMA/CA mit RTS/CTS (Ready To Send/Clear To Send) sowie die Point Coordination Function (PCF) zur Anwendung. Das bei einfachem CSMA/CA zur Anwendung kommende, in Abbildung 5 visualisierte Zugriffsverfahren wird als Distributed Coordination Function (DCF) bezeichnet. Hierbei hört eine sendewillige Station das Medium ab. Ist dieses frei, wird nach Ablauf des Zeitintervalls DCF Interframe Space (DIFS) der Sendevorgang eingeleitet (siehe Übermittlung des ersten Datenpakets von Station A an Station B).

Wird die Belegung des Mediums detektiert, wartet die Station, bis dieses wieder frei ist. Die sich dann anschließende Wartezeit vor Einleitung des Sendevorgangs setzt sich aus dem Zeitintervall DIFS und zufallsgeneriertem Backoff-Timer zusammen. Gegebenenfalls konkurrieren mehrere Stationen um das Medium (siehe Wettbewerb der Stationen A und C).

[19] Schneider/Werner: Taschenbuch der Informatik, S. 349.
[20] Rech: Wireless LANs, S. 15.
[21] Lewis: LAN-switching und Wireless, S. 465 ff.
[22] Enste/Müller: Datenkommunikation in der Prozessindustrie, S. 210.

Die Station, deren Backoff-Timer zuerst den Wert 0 annimmt, erhält in diesem Fall den Zugriff (siehe Übermittlung des zweiten Datenpakets von Station C an Station B). Die Wartezeiten der übrigen Stationen nach erneuter Feststellung eines freien Mediums setzen sich dann aus DIFS und dem letzten Stand des Backoff-Timers zusammen (siehe Übermittlung des dritten Datenpakets von Station A an Station B). Der Empfang von Datenpaketen wird generell nach Ablauf des Zeitintervalls SIFS (Short Interframe Space) mittels Acknowledge (ACK)-Kontrollframe bestätigt. Es gilt die Beziehung SIFS < DIFS.

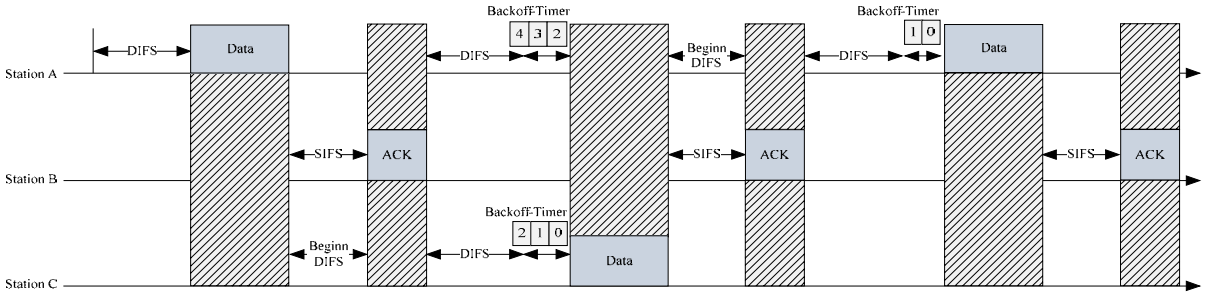


Abbildung 5 Einfaches CSMA/CA. Nach Abschluss des Sendevorgangs durch Station A konkurrieren die Stationen A und C um das Medium. Zunächst erhält C, dann A den Zugriff. Diese Abfolge wird durch den stochastisch ermittelten Backoff-Timer bestimmt. Empfänger ist in allen Fällen die Station B.

CSMA/CA mit RTS/CTS basiert ebenfalls auf der Distributed Coordination Function. Durch den RTS/CTS-Zusatz wird das Hidden Station Problem[23], das anhand von Abbildung 6 erläutert wird, gelöst. Gegeben seien vier Stationen A, B, C und D mit gleichen Sende- und Empfangsreichweiten. Für die Stationen B und C sind die Reichweiten mit Kreisen schematisch dargestellt. In der Beispielanordnung können die Stationen A (B) und C (D) auf B (C) zugreifen – jedoch befinden sich A (B) und C (D) außer Reichweite, sie sind voreinander „versteckt". Dies bedeutet, dass z. B. C (D) einen Sendevorgang von A (B) nach B (C) nicht detektieren kann. Bei Anwendung von einfachem CSMA/CA initiiert die Station C (D) also potentiell eine Datenübertragung nach B (C), was in der Konsequenz zu Signalüberlagerung führen würde.

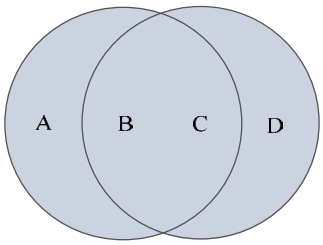


Abbildung 6 Das Hidden Station Problem beschreibt mögliche Signalüberlagerungen. Beispielsweise können A und C auf B zugreifen – jedoch ist A außerhalb der Reichweite von C. Ein Sendevorgang von A nach B ist für C daher nicht detektierbar. Folglich kann der Fall eintreten, dass A und C gleichzeitig nach B senden.

[23] Labiod/Afifi/Santis: Wi-Fi, Bluetooth, ZigBee and WiMax, S. 43.

Der Mechanismus CSMA/CA mit RTS/CTS, bei dem durch einen der Datenübertragung vorgelagerten Abstimmungsprozess Kollisionen infolge des Hidden Station Problems ausgeschlossen werden, ist in Abbildung 7 dargestellt. Jede Station in Reichweite der Sendestation empfängt die Sendedauer inklusive ACK-Kontrollframe über den initialen RTS-Frame und speichert diese im Net Allocation Vector (NAV). Die Bestätigung durch die empfangsbereite Station erfolgt über den CTS-Frame, der die verbleibende Sendedauer inklusive ACK-Kontrollframe beinhaltet. Hierdurch ist sichergestellt, dass auch alle Stationen in Reichweite der Empfangsstation über den Sendevorgang informiert sind und ihren NAV anpassen können. Das erste Datenpaket wird im Beispiel von Station A nach Station B übermittelt. Station C passt ihren NAV über den CTS-Frame der Station B an.

Analog dem Vorgehen bei einfachem CSMA/CA wird der Wettbewerb um das Übertragungsmedium mittels Backoff-Timer geführt. Das zweite Datenpaket wird im Beispiel von Station C nach Station B übertragen. Station D passt ihren NAV über den RTS-Frame der Station C an. Station A verwendet hierzu den CTS-Frame der Station B.

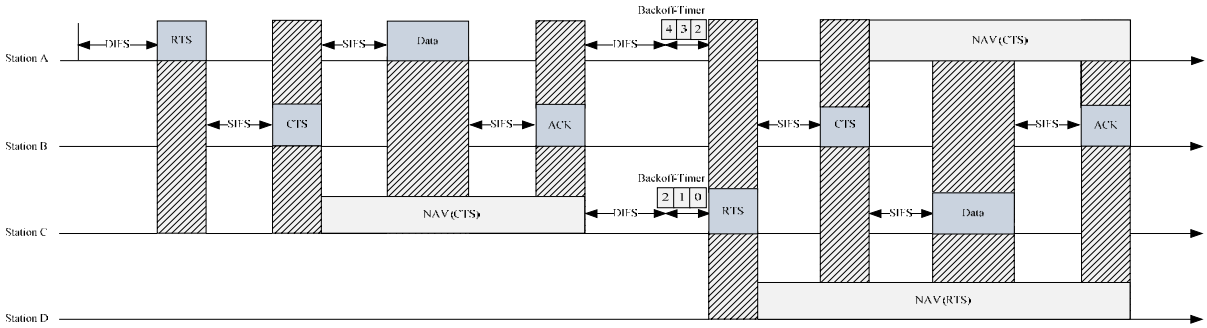


Abbildung 7 CSMA/CA mit RTS/CTS umgeht das Hidden Station Problem, indem vor dem Senden von Daten alle Stationen in Reichweite des Senders und Empfängers durch Setzen des jeweiligen Net Allocation Vector in einen passiven Zustand überführt werden. Im Beispiel sendet zunächst A nach B, C muss warten. Während der anschließenden Datenübermittlung von C nach B ruhen A und D.

Alternativ kann die in Abbildung 8 dargestellte Point Coordination Function zur Anwendung kommen. Bei diesem zentralen Zugriffsverfahren fungiert eine Station – in der Regel der Access Point – als koordinierende Instanz (KI). PCF wird aktiviert, indem die KI das Medium abhört und bei Feststellung der Nichtbelegung nach Ablauf des Zeitintervalls PIFS (PCF Interframe Space) ein den Start markierendes Kontrollframe (StartCF) schickt. Mit diesem wird die NAV-Zeit angegeben, für die die oben beschriebenen Backoff-Verfahren auszusetzen sind.

Nach Vergabe des Senderechts an eine Station durch ein Polling-Frame erfolgt dann die eigentliche Datenübertragung. Siehe hierzu die Übermittlung des ersten Datenpakets von Station B sowie die Übermittlung des zweiten Datenpakts von Station C. PCF wird durch ein das Ende markierendes Kontrollframe (EndCF) deaktiviert. Es gilt hier die Beziehung SIFS < PIFS < DIFS.

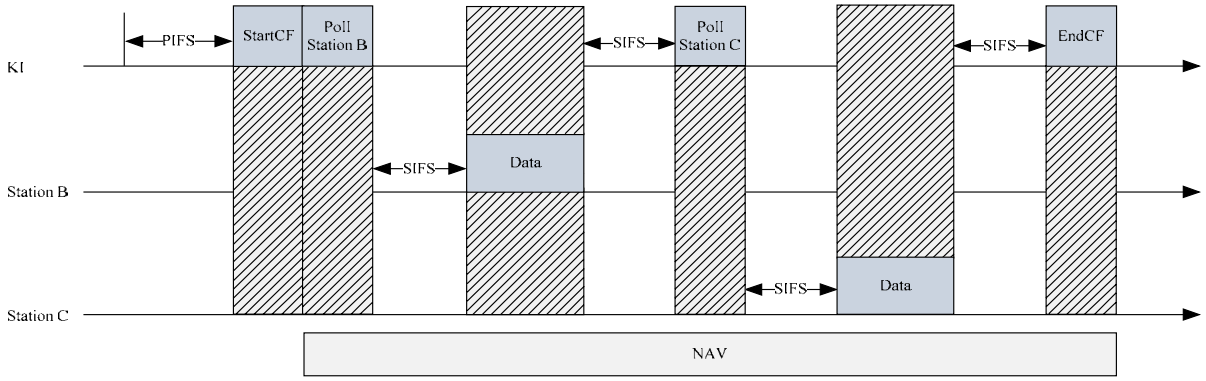


Abbildung 8 Point Coordination Function. Die koordinierende Instanz KI versetzt die Stationen B und C für eine im NAV zu speichernde Zeitdauer in einen Modus, der den eigenständigen Zugriff auf das Medium unterbindet. Im Beispiel weist die koordinierende Instanz zunächst explizit B, dann C an, vorhandene Daten zu übermitteln.

Mit Ausnahme von ZigBee können die Protokolle UDP, TCP und IP standardmäßig auf den höheren Schichten der beschriebenen WLAN- und WPAN-Technologien aufsetzen. Wie auch bei Ethernet stellt die obere Subschicht der Datensicherungsschicht (Logical Link Control) in Kombination mit Bridging das verbindende Element dar. Somit kann die Anbindung von WLAN/WPAN-basierten Netzwerken an das Internet vorgenommen werden (siehe Abbildung 9).

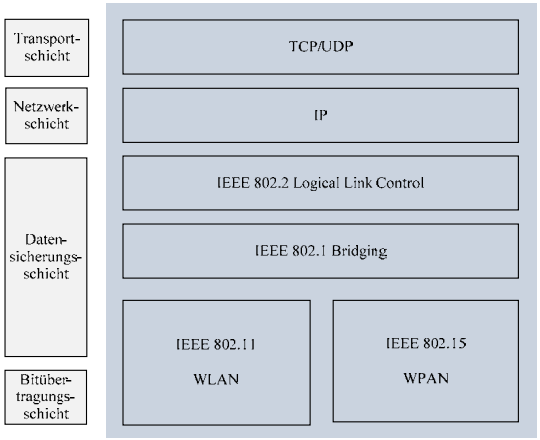


Abbildung 9 TCP/IP-Stack auf Grundlage von WLAN/WPAN. Die Spezifikationen der Datensicherungs- und Bitübertragungsschicht umfassen IEEE 802.2, IEEE 802.1, IEEE 802.11 sowie IEEE 802.15.

Der ZigBee-Stack hingegen greift direkt auf die MAC-Schnittstelle gemäß IEEE 802.15.4 zu. Hierdurch entfällt die IEEE 802.2 Logical Link Control-Schicht, was zu einer Reduktion des Implementierungsaufwands führt. Das von der ZigBee Alliance spezifizierte Protokoll der Netzwerkschicht wird als ZigBee Network Layer bezeichnet. Unmittelbar hierüber befindet sich das ZigBee Application Programming Interface[24]. Aufgrund der Substitution des Internet Protocol auf OSI-Schicht 3 ist ohne die Verwendung von Gateways die Anbindung von ZigBee-Geräten an das Internet nicht möglich[25].

[24] Sikora: ZigBee: Grundlagen und Applikation.
[25] Kupris/Sikora: ZigBee, S. 205.

Im Folgenden wird verdeutlicht, dass die drahtlose Kommunikation gemäß IEEE 802.11 die Forderung nach Sicherheit und Bedienkomfort gleichermaßen erfüllt. Zu übertragende Daten können optional unter Verwendung standardisierter Verschlüsselungsverfahren gesichert werden. WPA sowie dessen Nachfolgeversion WPA2 nehmen eine herausgehobene Stellung ein. Sie sind in IEEE 802.11i näher spezifiziert. WPA verwendet das Temporal Key Integrity Protocol (TKIP), WPA2 den Advanced Encryption Standard (AES)[26]. TKIP und AES zählen zur Kategorie der symmetrischen Kryptoverfahren, d. h. für die Chiffrierung und Dechiffrierung kommt derselbe Schlüssel zum Einsatz. Alle Kommunikationsteilnehmer müssen über diesen Schlüssel verfügen.

Prinzipiell existieren mehrere Möglichkeiten, die Schlüsselsynchronisation zu bewirken. Hierzu zählt die manuelle Konfiguration von Netzwerkgeräten. Vor dem Hintergrund, Nutzereingriffe auf ein Minimum zu beschränken, wurde der Standard Wi-Fi Protected Setup (WPS) – vormals als Wi-Fi Simple Config bezeichnet – entwickelt[27]. Hierbei entfällt die Notwendigkeit der manuellen Schlüsseleingabe. Stattdessen erfolgt der Schlüsselaustausch zwischen Access Point und Client nach Betätigen von deren – wahlweise in Hard- oder Software realisierten – Druckknöpfen. Insbesondere lassen sich auf diese Weise Geräte, die alphanumerische Eingaben nicht unterstützen, in eine gesicherte Umgebung einbringen. Alternativ ist vor dem Hinzufügen eines Gerätes zu einem geschützten Netzwerk eine eindeutige PIN einzugeben. Sie kann statisch vorgegeben oder dynamisch generiert werden.

2.1.3 Feldbussysteme

Im Gegensatz zu Ethernet kommen bei Feldbussystemen deterministische Zugriffsverfahren zum Einsatz. Damit sind Parameter wie Verzögerungszeiten und Jitter vorhersagbar. In Anwendungsdomänen mit Realzeitanforderungen bieten sich Feldbussysteme somit an. Deterministische Buszugriffsverfahren lassen sich in zentrale und dezentrale Verfahren kategorisieren. Für die Übertragung von Daten über einen Kanal kommen Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA) und Code Division Multiple Access (CDMA) zur Anwendung[28]. Zur Kategorie der zentralen Verfahren zählt die Master-Slave-Architektur, bei der die Buszuteilung durch eine Leitstation erfolgt. Im Bereich der Automatisierungstechnik dominiert TDMA. Zu nennen sind beispielsweise die Feldbussysteme ASI, Profibus, CC-Link und Sercos[29].

Zur Kategorie der dezentralen Verfahren zählen unter anderem Token Bus und CAN. Wesentliches Kennzeichen ist die Koordination des Buszugriffs durch die beteiligten Stationen. Beide Verfahren basieren auf TDMA. Bei Token Bus durchläuft eine Marke (Token) die in einer Ringstruktur angeordneten Stationen. Das Senderecht liegt ausschließlich bei der Station, die im Besitz der Marke ist.

CSMA/Collision Resolution (CSMA/CR), das die synonyme Bezeichnung CSMA/CA with Non Destructive Bit Arbitration (CSMA/CA NDBA) trägt, ist Grundlage von CAN. Durch die getroffene Wahl der Signalpegel ist die 0 mit einer Pegeldifferenz von 5 V dominant und die 1 mit einer Pegeldifferenz von 0 V rezessiv kodiert. Dies entspricht einer logischen UND-Verknüpfung. Mit Identifiern wird die Priorität von Nachrichten festgelegt. Je niedriger der Wert eines Identifiers ist, desto höher ist dessen Priorität.

---

[26] Lewis, W.: LAN-Switching und Wireless, S. 493.
[27] Carpenter, T.: Certified Wireless Technology Specialist Study Guide, S. 30 f.
[28] Keller: Datenübertragung im Kabelnetz, S. 83.
[29] Gevatter/Grünhaupt: Handbuch der Mess- und Automatisierungstechnik, S. 517 ff.

Leiten nun mehrere Stationen gleichzeitig den Sendevorgang ein, wird die weitere Nachrichtenübermittlung mittels Arbitrierung bestimmt. Hierzu wird der Übertragungskanal während der Übermittlung der Identifier von allen sendenden Stationen abgehört. Eine Station bricht den Sendevorgang ab, sobald der detektierte Signalpegel dem bisher gesendeten Identifier nicht mehr entspricht. Dies führt dazu, dass die Nachricht mit der höchsten Priorität versendet wird. Die Funktionsweise wird anhand von Abbildung 10 verdeutlicht. Im Beispiel gilt der Zusammenhang:

$$\text{Priorität(Station 3)} > \text{Priorität(Station 1)} > \text{Priorität(Station 2)}, \text{ da}$$
$$\text{Id(Station 3)} = 11001001101 < \text{Id(Station 1)} = 11001001111 < \text{Id(Station 2)} = 11001111111$$

Die Konkurrenzsituation wird demgemäß zugunsten von Station 3 aufgelöst. Es ist zu beachten, dass CSMA/CR nur als bedingt deterministisch betrachtet werden kann. Bei der Übermittlung niedrigpriorer Nachrichten sind signifikant schwankende Verzögerungszeiten zu erwarten. Verfahrensbedingt ist ausschließlich für die am höchsten priorisierte Nachricht eine definierte Antwortzeit zu garantieren.

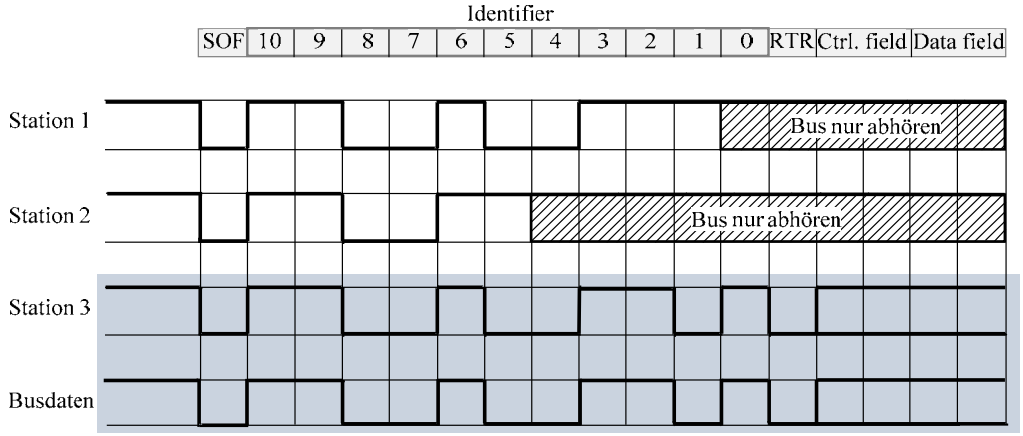


Abbildung 10 Wirkweise der Busarbitrierung bei CSMA/CR. Die Stationen 1,2 und 3 beginnen gleichzeitig den Sendevorgang. Station 3 setzt sich im Wettbewerb durch, da der zugehörige Nachrichtenidentifier die höchste Priorität hat.

Es ist davon auszugehen, dass klassische Heimnetzwerke hin zu Automatisierungslösungen konvergieren werden[30]. Ethernet-basierte Netze, die trotz CSMA/CD Echtzeitkommunikationsverhalten bieten werden unter dem Gattungsbegriff Industrial Ethernet zusammengefasst. Ein Ansatzpunkt ist der Übergang von Linien- zu Sterntopologien mit Switches. Bedingt geeignet sind Multiport-Repeater, die auch als Hubs bezeichnet werden. Bezogen auf den Informationsfluss führen diese nicht zu einer Auflösung der Linienstruktur. Mehrfachzugriffe auf das Übertragungsmedium sind demnach weiterhin möglich. Wird einschränkend eine reduzierte Netzlast vorausgesetzt, sinkt die Wahrscheinlichkeit von Kollisionen und das Verhalten wird quasi-deterministisch. Der beschriebene Zusammenhang ist in Abbildung 11 qualitativ wiedergegeben[31]. Die durchschnittliche Verzögerung ist hierbei normiert.

[30] Gevatter/Grünhaupt: Handbuch der Mess- und Automatisierungstechnik, S. 597.
[31] Furrer: Industrieautomation mit Ethernet-TCP/IP und Web-Technologie, S. 64.

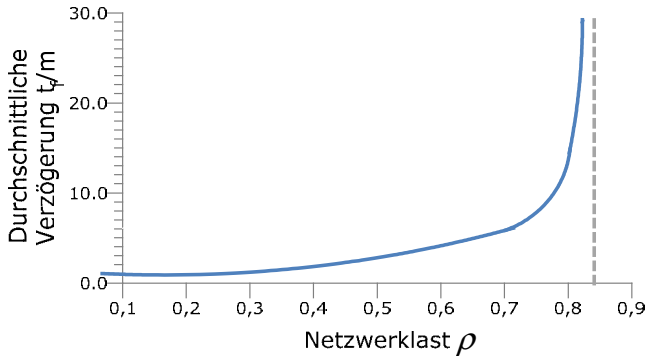


Abbildung 11 Ethernet Zeitverzögerungen in Abhängigkeit der Netzwerklast. Mit zunehmender Netzwerklast steigt die Wahrscheinlichkeit von Kollisionen. In der Folge ist ein Anstieg der durchschnittlichen Verzögerung zu verzeichnen.

Für die konkrete Dimensionierung von Netzwerken ist ebenfalls der zeitliche Verlauf der Netzwerklast von Relevanz. Diesbezüglich wurde nachgewiesen, dass der Datenverkehr in praxi eine selbstähnliche Struktur aufweist[32]. Hinzuweisen ist insbesondere auf die Nichteignung von Modellen, die den fraktalen Charakter unbeachtet lassen. In diesem Zusammenhang zu nennen sind beispielsweise Modelle, bei denen von einer Poisson-verteilten Meldungsgenerierung ausgegangen wird[33,34].

Mit dem Einsatz von Store-and-Forward-Switches können Teilnehmer für die Dauer der Kommunikation paarweise miteinander verbunden und Kollisionen somit ausgeschlossen werden. Die Speicherung gegebenenfalls anfallender Datenpakete von nicht durchgeschalteten Teilnehmern erfolgt in den Switches.

Das zeitlich exakt modellierbare Warteschlangenverhalten trägt zu definierten Zugriffszeiten in Sterntopologien bei. Durch die priorisierte Behandlung von Datenpaketen können Dienste darüber hinaus gemäß ihren Echtzeiterfordernissen zugeordnet werden. Dies ermöglicht z. B. den Betrieb zeit-unkritischer und zeitkritischer Dienste über eine Infrastruktur. Die Priorisierung kann durch Bildung virtueller Netzwerksegmente auf unterschiedlichen Ebenen des ISO/OSI-Schichtenmodells erfolgen. Diese Virtual LANs (VLANs) sind beispielsweise über MAC-Adressen oder Kennungen in der Anwendungsschicht bestimmt.

Die ausschließliche Verwendung von Switches als Verbindungselement zwischen Stationen stellt ein Extremum dar. Innerhalb von Kollisionsdomänen, die durch Zusammenfassen von Stationsgruppen gebildet werden, kann bei schwächeren Echtzeitanforderungen auf Switches verzichtet werden. Dies liegt darin begründet, dass bei geeigneter Segmentierung nur geringe Verzögerungen zu erwarten sind und es somit vielfach ausreichend ist, die jeweilige Kollisionsdomäne an einen Switch anzubinden.

Eine weitere Alternative stellt der direkte Eingriff in die Bussteuerung dar, um auf diese Weise eine zentrale Organisation mit einem Manager und mehreren Controllern zu realisieren. Dieser Ansatz liegt z. B. Powerlink zugrunde[35]. Hier wird das zentralistische Kommunikationsverhalten über den Powerlink-Protokollstapel implementiert.

---

[32] Leland et al.: On the Self-Similar Nature of Ethernet Traffic.

[33] Hammond/Peter: Performance Analysis of Local Computer Networks.

[34] Schwartz: Telecommunication Networks, S. 455 f.

[35] Schnell/Wiedemann: Bussysteme in der Automatisierungs- und Prozesstechnik, S. 332 ff.

Durch geeignete Einbettung des TCP/IP-Protokollstapels können Applikationen unmittelbar hierauf zugreifen (siehe Abbildung 12). Im OSI-Modell ist Powerlink auf den Schichten 2 bis 7 angesiedelt. Damit ist es von der zur Anwendung kommenden Bitübertragungsschicht unabhängig. Von Praxisrelevanz sind jedoch ausschließlich Protokollstapel mit Ethernet als Unterbau.

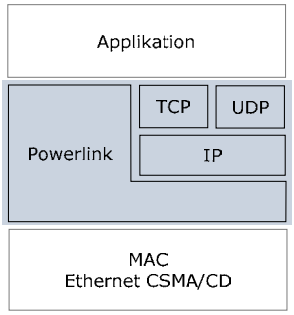


Abbildung 12 Der Powerlink-Protokollstapel ist oberhalb von Ethernet angeordnet. Er verfügt über proprietäre Funktionen für die Echtzeitkommunikation und bindet TCP/IP zur parallelen Nutzung durch Applikationen mit ein.

Werden TCP/IP-fähige Endgeräte ohne Powerlink-Fähigkeit in eine Powerlink-Umgebung eingebunden, ist ein Powerlink-Switch dazwischen zu schalten. So ist sichergestellt, dass in der Powerlink-Domäne weiterhin eine kollisionsfreie Kommunikation möglich ist. Umgekehrt können Powerlink-Geräte ohne Weiteres in einer Nicht-Powerlink-Umgebung zum Einsatz kommen. Auf diese Weise wird Kompatibilität gewährleistet, allerdings unter Umgehung des zentralistischen Verfahrens.

Zu den gängigen industriellen Ethernet-Varianten zählen Modbus/TCP, Ethernet/IP, HSE, Powerlink, Profinet, Sercos III und EtherCAT[36]. Die beiden zuletzt genannten Varianten weisen maximale Jitter zwischen 0,5 µs und 15 µs auf. Sie erfüllen damit die Anforderungen der höchsten Echtzeitklasse der Industrial Automation Open Networking Alliance (IAONA).

[36] Gevatter/Grünhaupt: Handbuch der Mess- und Automatisierungstechnik, S. 609.

2.1.4 KNX/EIB

KNX ist ein offener Standard für die Gebäudeautomatisierung, der durch die KONNEX Association begründet wurde. Er stellt eine Weiterentwicklung des Europäischen Installationsbusses (EIB) dar, zu dem Abwärtskompatibilität besteht. KNX unterstützt die Übertragungsmedien Twisted Pair, Powerline, Funk sowie Ethernet.

Wird die Betrachtung auf die Datenübertragung reduziert, bietet KNX gegenüber den weiter oben diskutierten Standards Ethernet bzw. IEEE 802.11 keinen Zusatznutzen. Dieser ergibt sich erst aus der Möglichkeit, mittels KNX ergänzend die Geräteverwaltung und -steuerung vornehmen zu können. Mit Blick auf die Zielstellung der vorliegenden Arbeit eignen sich die zugrunde liegenden Mechanismen jedoch nicht. Diesbezüglich sprechen im Wesentlichen zwei Gründe gegen KNX.

Zum einen ist der Gerätenutzung eine aufwändige Installationsphase vorgelagert, die typischerweise das Hinzuziehen qualifizierter Elektroinstallateure erfordert. Im diesem Zusammenhang erfolgt beispielsweise die Einordnung eines Gerätes in den hierarchisch strukturierten Gesamtkomplex der Gebäudeinstallationsanlage mitsamt der Definition von Sensor-Aktor-Beziehungen. Die Angaben schaffen die Basis für den späteren Informationsaustausch zwischen Sensor und Aktor mittels sogenannter Kommunikationsobjekte. Diese stellen synchronisierbare Speicherbereiche dar. Abbildung 13 zeigt ein Beispiel, bei dem gleichnamige Kommunikationsobjekte von Sensor und Aktor miteinander verknüpft sind. Ändert sich das Kommunikationsobjekt des Sensors infolge der aktuellen Signalwertaufnahme, wird auch das entsprechende Kommunikationsobjekt des Aktors über eine Telegrammmitteilung aktualisiert. Deren Empfang wird wiederum vom Aktor mit einem Bestätigungstelegramm quittiert[37].

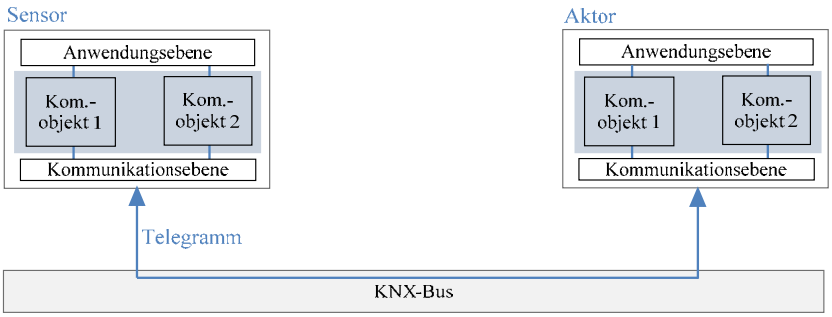


Abbildung 13 Informationsaustausch zwischen Aktor und Sensor in einer KNX-Umgebung. Die Grundlage bilden hierbei synchronisierbare Speicherbereiche, sogenannte Kommunikationsobjekte. Ändert sich ein Kommunikationsobjekt des Sensors führt dies durch Versand eines Telegramms zur Aktualisierung des assoziierten Kommunikationsobjektes des Aktors.

Im Weiteren umfasst die Installationsphase die Geräteparametrierung. Hier kann z. B. ein Taster als Dimmer oder alternativ als Ein-Aus-Schalter klassifiziert werden. Erstreckt sich die Parametrierung über einen großen Funktionsumfang ist hierzu der sogenannte System Mode, auf den gegenwärtig ca. 85 % aller KNX-Geräte ausgelegt sind, zu wählen[38]. Dieser bedingt den Einsatz eines speziellen PC-Werkzeugs wie der Engineering-Tool-Software (ETS). Selbiges trifft auf alle weiteren hier beschriebenen Einstellungen zu.

[37] Merz/Hansemann/Hübner: Gebäudeautomation, S. 111 ff.
[38] Kriesel/Sokollik/Helm: KNX/EIB für die Gebäudesystemtechnik in Wohn- und Zweckbau, S. 19.

KNX ist an einer weiteren zentralen Stelle mit dem im Rahmen der Arbeit verfolgten Plug-and-Play-Gedanken unvereinbar. KNX-basierte User Interfaces für Steuergeräte wie z. B. Touchpanels sind statischer Natur. Im Vorfeld ist festzulegen, in welcher Form bestimmte Gerätearten dargestellt werden.

Die mangelnde Fähigkeit der Autogenerierung von User Interfaces liegt darin begründet, dass KNX-Geräte keine Informationen über die zwischen Funktionen und Gerätezuständen bestehenden Abhängigkeiten liefern. Diese gehen vielmehr aus der jeweiligen Gerätespezifikation hervor. In der Konsequenz muss daher bei Aufnahme einer neuen Geräteart dessen Modell programmtechnisch abgebildet werden. Damit eignet sich KNX hauptsächlich für Anwendungsbereiche, bei denen die Steuerung einer begrenzten Auswahl von spezifizierten Produkten vorgesehen ist, weniger als Grundlage eines breiter gefassten Gerätespektrums.

2.2 Verteilte Ad-Hoc-Systeme

Es ist festzustellen, dass die formale Beschreibung der Syntax und Semantik der internetbasierten Kommunikation zahlreicher Geräte gegenwärtig über proprietäre Protokolle erfolgt. Diesen ist in der Regel ausschließlich gemein, dass sie auf dem TCP/IP-Protokollstapel aufsetzen. Eine unmittelbare Implikation der Protokollvielfalt ist die Notwendigkeit, produktrespektive herstellerspezifische Steuersoftware einzusetzen. In der Folge bietet sich dem Anwender ein inhomogenes Erscheinungsbild im Hinblick auf die von ihm genutzten Schnittstellen. Ein weiteres Defizit heterogener und zueinander inkompatibler Systeme ist die mangelnde Fähigkeit der Interkommunikation.

Des Weiteren ist die Netzwerkanbindung in vielen Fällen mit manuellen Eingriffen bei der Gerätekonfiguration verbunden. Bereits heute sind Infrastrukturtechnologien verfügbar, bei denen Netzwerkverbünde ohne aktive Einbindung des Anwenders dynamisch gebildet werden. Zu den marktgängigen Technologien zählen Jini, Zeroconf, UPnP und IEEE 1451. Mit dem Ziel, diese unter dem Aspekt des Implementierungsaufwandes abzuschätzen, erfolgt eine eingehende Analyse wesentlicher Bestandteile der zugrunde liegenden Protokolle.

Die dienstorientierten Ad-Hoc-Systeme Jini, Zeroconf und UPnP ähneln sich in vielerlei Hinsicht. Diesbezüglich wird durchgängig differenziert in Protokolle für die Bekanntgabe von Diensten, die Suche nach Diensten, Statusupdates sowie die Nutzung von Diensten. Für den Bezug von IP-Adressen eignet sich der kombinierte Einsatz von DHCP und Auto-IP auf die im Folgenden beschriebene Weise.

Nach dem Anschluss eines Gerätes an den Netzwerkverbund wird zunächst der Versuch unternommen, über DHCP eine gültige IP-Adresse zu erlangen. Ist eine Adresszuordnung über einen DHCP-Server nicht möglich, so wird mittels Auto-IP eine im lokalen Kontext gültige IP-Adresse („non routable“) bezogen. Diese entstammt dem zweiten der gemäß RFC 3330[39] für die private Nutzung reservierten vier Adressbereiche, d. h. 169.254.0.0 bis 169.254.255.255. Es handelt sich um den in RFC 3927[40] spezifizierten Link-Local-Adressbereich. Nach Auswahl einer Link-Local IP-Adresse wird das originär für die Zuordnung von IP- zu MAC-Adressen entwickelte Address Resolution Protocol (ARP) genutzt, um Adresskonflikte auszuschließen. Hierzu muss das ‚target IP'-Datenfeld der ARP-Anfrage die ausgewählte IP-Adresse beinhalten. Bei Ausbleiben einer ARP-Antwort über ein mehrere Sekunden andauerndes Zeitintervall ist die ausgewählte IP-Adresse noch nicht zugeordnet und kann somit verwendet werden.

[39] IANA: Special-Use IPv4 Addresses, RFC 3330.
[40] CHESIRE/ABOBA/GUTTMAN: Dynamic Configuration of IPv4 Link-Local Addresses, RFC 3927.

Wird hingegen eine ARP-Antwort empfangen, so deutet dies auf eine bereits erfolgte Belegung hin. In diesem Fall muss eine neue IP-Adresse ausgewählt werden und die beschriebenen Schritte zur Adresskonfliktvermeidung sind zu wiederholen.

2.2.1 Jini

Die auf der Programmiersprache Java basierende Jini-Technologie wurde im Jahr 1998 durch Sun Microsystems eingeführt. Sie ermöglicht die Ad-Hoc-Kommunikation in verteilten Systemen. Mit dem Protokoll Join werden Dienste bekanntgegeben. Für die Suche nach Diensten bilden die Protokolle Discovery und Lookup die Grundlage[41]. Hierbei fungiert ein Lookup-Server als Mittler. Weiter unten folgt die Darstellung einer beispielhaften Kommunikation im zeitlichen Verlauf. Protokolle für Statusupdates sowie die Nutzung von Diensten unterliegen nicht der Spezifikation. Hierfür eignet sich u. a. RMI (Remote Method Invocation), das Bestandteil von Jini ist. Ebenso können alternative Protokolle eingesetzt werden.

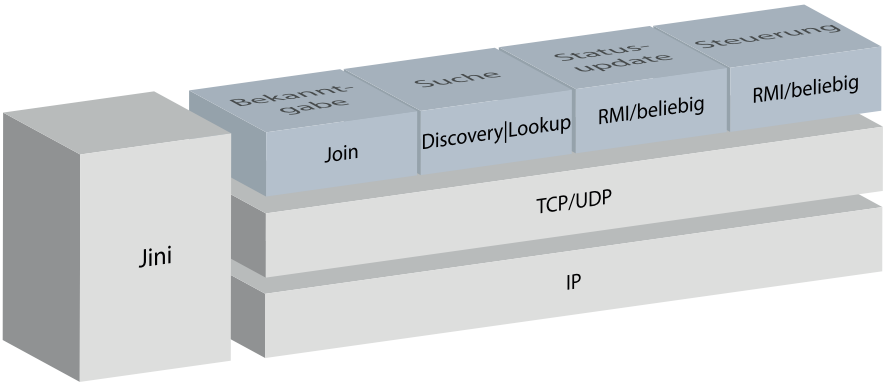


Abbildung 14 Protokollstapel von Jini. Dienste werden mittels Join-Protokoll bei einem Lookup-Server bekanntgegeben. Für die Suche hiernach steht das Lookup-Protokoll zur Verfügung. Die entsprechenden Anfragen sind direkt an den Dienste verwaltenden Lookup-Server zu richten. Dieser ist im Vorfeld unter Nutzung des Discovery-Protokolls zu ermitteln. Da RMI integraler Bestandteil von Jini ist, bietet es sich für Statusupdates sowie die Gerätesteuerung an. Wahlweise können andere Protokolle verwendet werden.

Der initiale Vorgang für die Bekanntgabe des mit einem Jini-fähigen Gerätes assoziierten Dienstes ist die Suche eines Lookup-Services mittels Discovey Protocol. Nach Anschluss des Jini-Gerätes *J1* durch *Aktor 1* erfolgt die Ermittlung und der anschließende Download des für die Diensteregistrierung benötigten Lookup-Service-Proxys durch den Dienst (1-3). Im Rahmen der Diensteregistrierung mittels Join Protocol wird der Upload des Service Proxys an den Lookup-Service vollzogen (4). Mit Abschluss dieses Schrittes ist der Dienst integraler Bestandteil des Jini-Verbundes und somit *Aktor 2* über einen ihm bereitstehenden Client zugänglich (5). Hierzu muss der Client zunächst den für die Dienstesuche erforderlichen Lookup-Service mittels Discovery Protocol ermitteln und den zugehörigen Lookup-Service-Proxy herunterladen (6–7). Unter Nutzung dieses Lookup-Service-Proxys wird der Client befähigt, einen Dienst gemäß vorgegebenem Suchmuster mittels Lookup Protocol zu identifizieren und den für seine Kontrolle zu nutzenden Service-Proxy herunterzuladen (8–9). Die Steuerung des Jini-Gerätes *J1* durch *Aktor 2* wird mittels Client vorgenommen (10).

[41] Arnold et al.: The Jini Specification, S. 83 ff.

Die Funktionalität des Dienstes kann über den Service-Proxy in Anspruch genommen werden (11). Hierbei kann ein frei wählbares Protokoll eingesetzt werden. Vom Service an den Client zurückgelieferte Ergebnisse werden für die weitere Interaktion anwendergerecht aufbereitet (12–13). Der beschriebene Kommunikationsprozess ist in Abbildung 15 dargestellt.

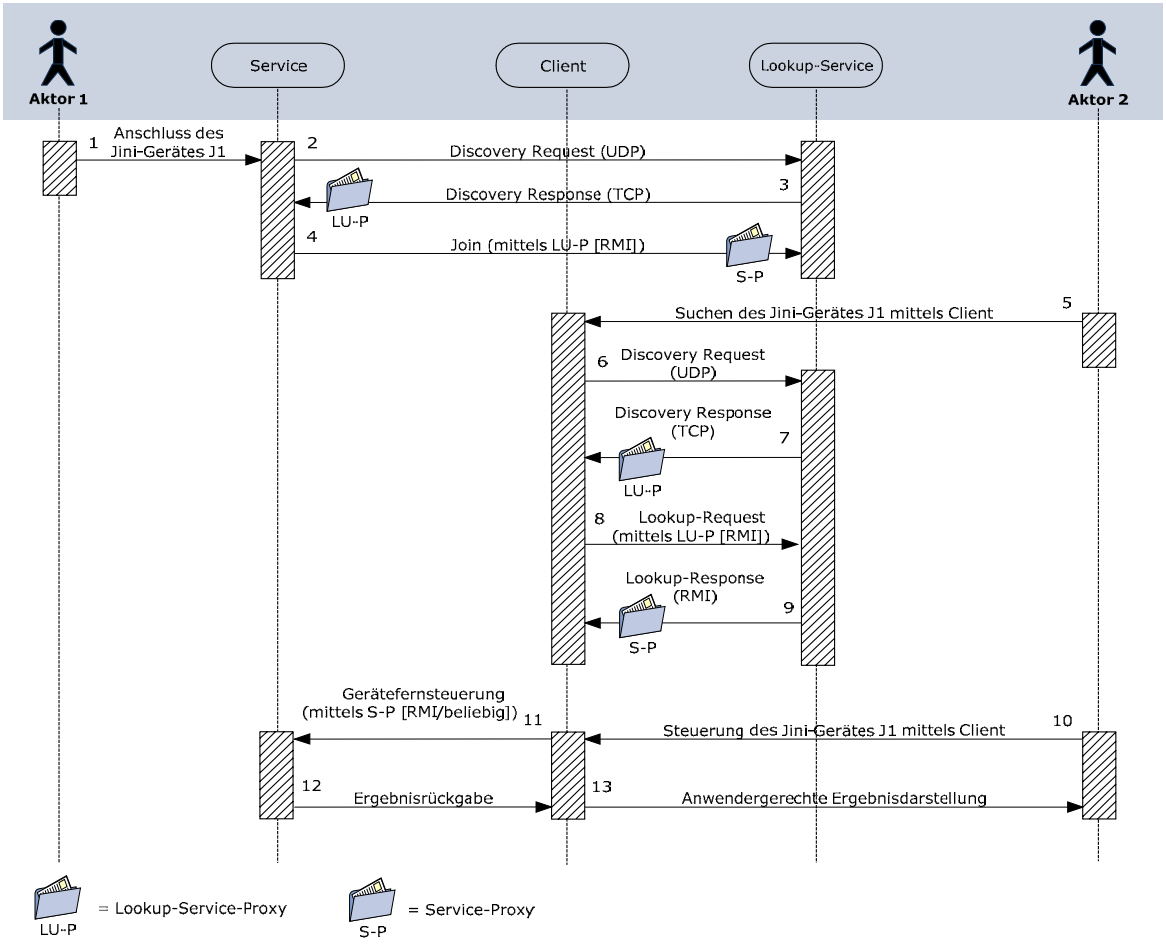


Abbildung 15 Beispielhafte Kommunikation im Jini-Verbund. *Aktor 1* schließt das Gerät *J1*, das sich beim Lookup-Service registriert, an. Darauf folgend nutzt *Aktor 2* einen Client, um *J1* unter Rückgriff auf den Lookup-Service aufzufinden. Die Steuerung von *J1* schließt sich an.

Die Funktionalität von Diensten wird wie bereits beschrieben über Proxies bereitgestellt, die das Konzept der verteilten Programmierung realisieren. Erfolgt der Aufruf von Proxyfunktionen durch einen lokalen Client, wird der zugehörige Programmcode auf dem Remoterechner ausgeführt und das Ergebnis zurück an die aufrufende Instanz übermittelt. Bezüglich des dem Service-Proxy zugrunde liegenden Protokolls bestehen keine Einschränkungen.

In der Regel werden Service-Proxies mittels RMI realisiert, da dieses Protokoll integraler Bestandteil der Java Standardbibliotheken ist. Gemäß Jini Device Architecture Specification[42] erfolgt die Kommunikation von Lookup-Service-Proxies ausschließlich über RMI. Im zeitlichen Ablauf stellt sich die Funktionsweise von RMI wie im Folgenden beschrieben dar. Zunächst registriert der Server unter Angabe eines Binding-Namens bei seinem Start ein Objekt mit einer für den Remote-Zugriff spezifizierten Schnittstelle.

[42] Sun Microsystems: Jini Device Architecture Specification.

Die Kombination aus der Zeichenkette „rmi://“, der IP-Adresse des Servers und des Binding-Namens bildet im Kommunikationsprozess die vom Client benötigte eindeutige Kennung der Serverinstanz. Der clientseitig zu initiierende Methodenfernaufruf bedingt nun die vorherige Ermittlung des so genannten Stubs unter Verwendung der oben beschriebenen Kennung der Serverinstanz. Das Stub-Objekt implementiert hierbei dieselbe Schnittstellenspezifikation wie das Server-Objekt und leitet die vom Client durchgeführten Methodenaufruf an das serverseitig befindliche Skeleton weiter.

Das Skeleton wiederum stellt sicher, dass die entsprechende Methode des Server-Objektes ausgeführt wird. Ergebnisse gehen über denselben Weg in umgekehrter Reihenfolge zurück. Bei der Anwendungsentwicklung werden die erforderlichen Client-Stub- und Server-Skeleton-Klassen ab Java-Version 5 automatisch generiert. Die explizite Klassengenerierung mittels des Programms *rmic* – wie dies in früheren Java-Versionen notwendig war – entfällt[43]. Abbildung 16 visualisiert das Prinzip des Methodenfernaufrufs mittels RMI.

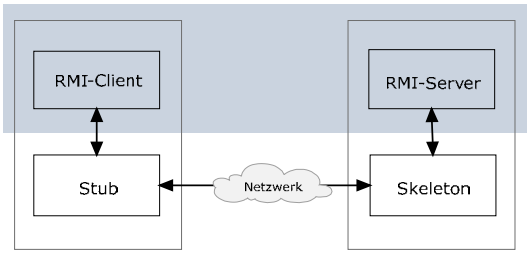


Abbildung 16 Prinzip des RMI-basierten Methodenfernaufrufs. Der Client greift auf das Stub-Objekt zu, das Remote-Aufrufe an das serverseitig befindliche Skeleton-Objekt weiterleitet. Auf dem umgekehrten Weg werden Ergebnisse zurückgeliefert.

Der Zugriff auf Jini-Dienste kann über die Definition von Zugangskontrolllisten gesteuert werden. Neben dem Erteilen und Verwehren benutzerspezifischer Zugriffsrechte bietet Jini den Sicherheitsmechanismus der verschlüsselten Datenübertragung wie z. B. die SSL-basierte RMI-Kommunikation[44]. Es ist demnach aufwandsarm möglich, das Grundgerüst für verteilte Anwendungen mit frei verfügbaren Java-Entwicklungsumgebungen zu erstellen.

Das Ausführen von Jini-Diensten setzt eine Java Runtime Environment (JRE) mit RMI sowohl auf dem steuernden als auch zu steuernden Gerät voraus. Der hierfür benötigte Speicherplatz beläuft sich auf ca. 20 MB. Mit dem Ziel, den Speicherplatzbedarf zu minimieren, ist alternativ der Einsatz einer spezialisierten JRE, die ausschließlich den vom Gerät benötigten Funktionsumfang implementiert, möglich. Derartige Individuallösungen sind jedoch mit einem hohen Entwicklungsaufwand verbunden.

Typische Konstellationen, bei denen die JRE des Dienste-anbietenden Gerätes ausgelagert wird, sind die Netzwerkoption, die physikalische Option und Jini über IIOP. Bei der Netzwerkoption (Network Option) fungiert ein Gateway als Vermittlungseinheit, die über proprietäre Protokolle mit dem Dienste anbietenden Gerät und über ein frei wählbares Protokoll – in der Regel RMI – mit dem Dienste nutzenden Gerät kommuniziert. Somit kann der benötigte Bibliotheksumfang in dem Dienste anbietenden Gerät erheblich reduziert werden. In dieser Anordnung ist zusätzlich ein Netzwerk-Gateway erforderlich. Siehe hierzu Abbildung 17.

[43] Abts: Masterkurs Client/Server-Programmierung mit Java, S. 235 ff.
[44] Bader/Huber: Jini – Die intelligente Netzwerkarchitektur in Theorie und Praxis.

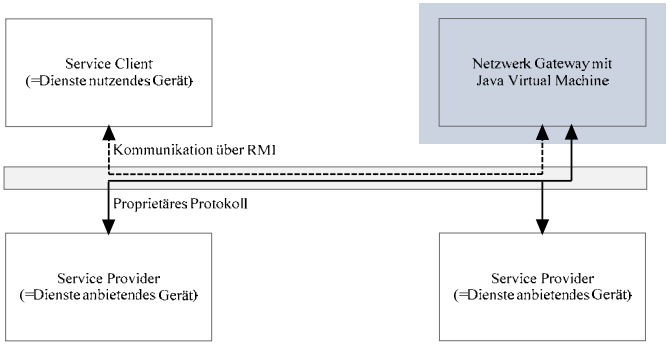


Abbildung 17 Jini Netzwerkoption. Ein Netzwerk-Gateway kommuniziert über ein proprietäres Protokoll mit Service Provider und über RMI mit Service Clients. Durch den Mechanismus der Protokollumsetzung kann der Bibliotheksumfang auf dem Dienste anbietenden Gerät reduziert werden.

Bei der physikalischen Option (Physical Option) stellt die Device Bay das Analogon zum Netzwerkproxy dar mit dem Unterschied, dass eine direkte Hardwareverbindung gegeben ist (siehe Abbildung 18). Die Device Bay kommuniziert in diesem Fall ebenfalls über ein proprietäres Protokoll mit dem Dienste anbietenden Gerät und über Jini-Basisprotokolle mit dem Dienste nutzenden Gerät. Beiden Optionen ist gemein, dass der Betrieb nicht-Java-fähiger Geräte spezifische Netzwerkstrukturen voraussetzt. Dieser Umstand ist nicht mit dem Grundgedanken ortsungebundener Dienstebereitstellung in sich durch mobile Endgeräte konstituierenden Ad-Hoc-Netzwerken in Einklang zu bringen.

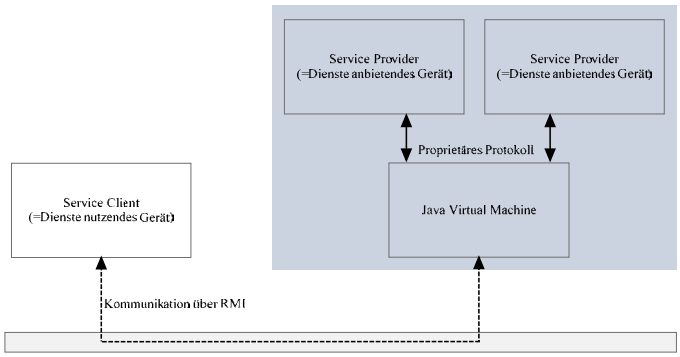


Abbildung 18 Physikalische Option. Service Provider sind direkt mit einer Device Bay verbunden. Diese verfügt über eine Java Virtual Machine und erlaubt somit die RMI-basierte Kommunikation mit Service Clients.

Eine weitere Alternative zu rein Java-basierten Lösungen besteht in der Nutzung des Internet Inter-Operability Protocol (IIOP), das durch CORBA (Common Object Request Broker Architecture) ORB (Object Request Broker) unterstützt wird. Bei CORBA handelt es sich um eine programmiersprachenunabhängige Spezifikation der OMG (Object Management Group), die den Methodenaufruf verteilter Objekte beschreibt. Dieser Ansatz setzt voraus, dass der Jini Lookup-Dienst sowohl das native RMI als auch IIOP unterstützt. Mit Einsatz von RMI over IIOP wird dieser Forderung nachgekommen. Dienste anbietende Geräte, die ein CORBA ORB beinhalten, benötigen die Implementierung der Lease-Klasse des Lookup-Dienstes, um so beispielsweise ihre Gültigkeitsdauer zu verlängern.

Eine erhebliche Einschränkung der Geräteinteraktion ergibt sich aus der Tatsache, dass der nicht-Java basierten Instanz des Service Providers der Zugriff auf Proxies Jini-kompatibler Dienste (z. B. Proxies von Lookup-Services) verwehrt ist. Die Funktionalität Jini-kompatibler Geräte ist also per se mittels CORBA ORB nicht zugänglich. Da die Implementierung des Proxies eines Dienste anbietenden Gerätes in Java vorgenommen werden muss, erfolgt dessen Registrierung beim Lookup Service konventionell mittels RMI.

Abbildung 19 ist zu entnehmen, dass in diesen Schritt die Java basierte Instanz eingebunden ist. Ein Dienste nutzendes Gerät kann diesen Proxy dann von dem Lookup Service mittels RMI herunterladen, um steuernd auf die nicht-Java basierte Instanz mit einem frei wählbaren Protokoll zuzugreifen. Wie bereits weiter oben ausgeführt, dient die Kommunikation mittels IIOP zwischen der nicht-Java basierten Instanz und dem Lookup Service z. B. der Verlängerung der Gültigkeitsdauer des jeweiligen Dienstes. Die beiden beschriebenen Instanzen des Service Providers können räumlich voneinander getrennt sein.

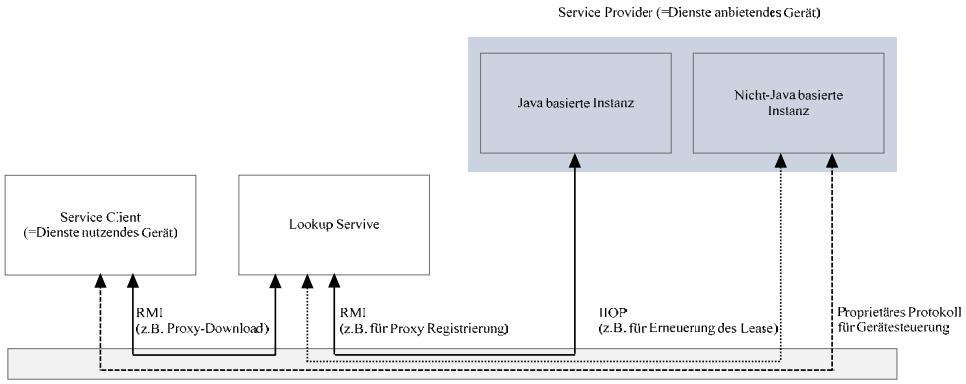


Abbildung 19 Jini und IIOP. Service Provider nutzen RMI ausschließlich für die Proxy Registrierung. Hierzu wird eine Java-basierte Instanz, die gegebenenfalls ausgelagert ist, benötigt. Im weiteren Verlauf – z. B. für die Erneuerung des Lease – kommt IIOP zum Einsatz. Die Steuerung des Dienste anbietenden Gerätes durch einen Service Client erfolgt über ein frei wählbares Protokoll.

Abbildung 20 stellt die Klassifikation von Geräten im Jini-Verbund gemäß Jini Device Architecture Specification in der Übersicht dar. Wie weiter oben geschildert, sind alle fünf Varianten mit Vor- und Nachteilen verbunden. Im Ergebnis vermag keine zu überzeugen.

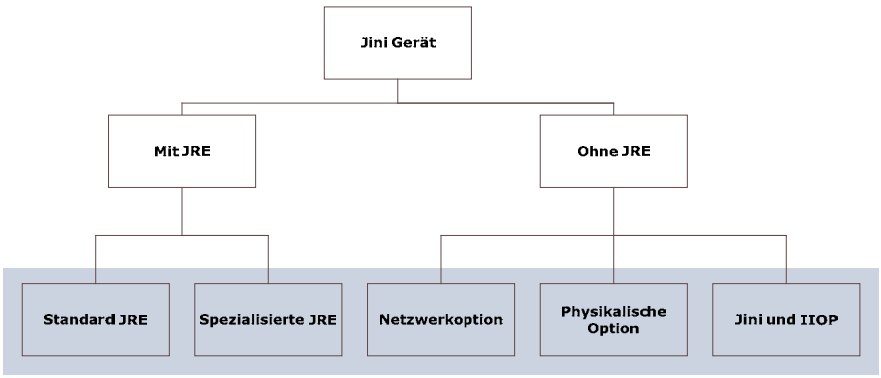


Abbildung 20 Klassifikation von Geräten im Jini-Verbund. Bei den fünf Varianten wird unterschieden in Geräte mit und ohne Java Runtime Environment.

In den letzten Jahren konnte sich Jini am Markt nicht durchsetzen. Bis heute ist kein kommerziell erfolgreiches Jini-Produkt verfügbar. Mit Jini ist es nicht gelungen, grundlegende Anforderungen an eine intelligente Netzwerkinfrastruktur bei gleichzeitig geringem Ressourcenbedarf zu erfüllen.

2.2.2 Zeroconf

Im Jahr 2002 wurde die Zeroconf-Technologie unter der Produktbezeichnung „Rendezvous“ durch Apple eingeführt. Aus rechtlichen Gründen erfolgte mit der Einführung von Mac OS X v10.4 im Jahr 2005 eine Umbenennung in „Bonjour“[45]. Mit ihr wird eine Umgebung bereitgestellt, die keiner aufwändigen Nutzerinteraktion für die Netzwerkkonfiguration bedarf.

Das DNS-Protokoll sowie dessen Multicast-Derivat mDNS finden Verwendung für die Bekanntgabe von Diensten, die Suche nach Diensten und Statusupdates[46]. DNS-SD lässt sich über lokale Netzwerkgrenzen hinweg einsetzen. Die prinzipielle Funktionsweise wird im Weiteren anhand des lokalen Anwendungsfalls erläutert. Das Protokoll für die Nutzung der Dienste ist frei wählbar. Im unten vorgestellten Beispiel ist es HTTP. Abbildung 21 stellt die beschriebenen Zusammenhänge dar.

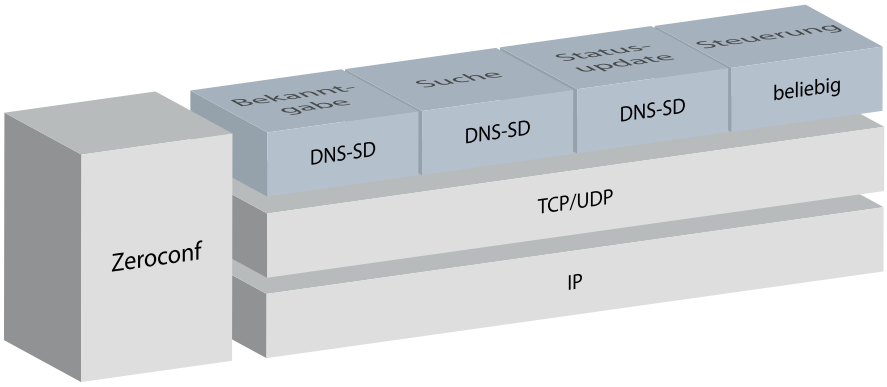


Abbildung 21 Protokollstapel von Zeroconf. Die Gerätebekanntgabe und -suche werden genauso wie Statusupdates auf Grundlage von DNS-SD vollführt. Für die Gerätesteuerung kann ein frei bestimmbares Protokoll genutzt werden.

Ein dem Netzwerk hinzugefügtes Geräte verschickt mDNS-Nachrichten, um die mit dem Gerät assoziierten Dienste allen dem Verbund angehörigen Clients bekanntzugeben. Darüber hinaus können mDNS-Nachrichten durch Clients für die zielgerichtete Suche nach Diensten unter Angabe von näher beschreibenden Attributen genutzt werden.

Zeroconf stützt sich auf eine Reihe von RFCs, von denen die für das Grundverständnis relevanten im Weiteren explizit Erwähnung finden. Der Adressbezug erfolgt wie bereits in Kapitel 2.2 ausgeführt über die in den RFCs 2131[47] und 826[48] beschriebenen Protokolle DHCP sowie ARP.

[45] Kobylinski: Max OS X Tiger, S. 46.
[46] Chesire/Steinberg: Zero Configuration Networking, S. 32 ff.
[47] Droms: Dynamic Host Configuration Protocol, RFC 2131.
[48] Plummer: An Ethernet Address Resolution Protocol, RFC 826.

Auf die Verfahrensweise des dynamischen Bezugs von Link-Local-Adressen gemäß RFC 1918[49] unter Verwendung von ARP wird in RFC 3927[50] vertiefend eingegangen. mDNS-Nachrichten basieren auf der in RFC 1035[51] definierten Nachrichtenstruktur bestehend aus einem Header, Anfrage-, Antwort-, Authority- und Zusatzbereich. Der Header beschreibt die Art und Struktur einer Nachricht. Die Nutzung der Datenbereiche, in denen Informationen in Form von Resource Records codiert werden, ist eng mit der Nachrichtenart korreliert.

Für den Versand von mDNS-Nachrichten wird die Multicast-Adresse 224.0.0.251, Port 5353 verwendet. Informationen innerhalb von mDNS-Nachrichten sind in Form von Resource Records strukturiert. Zu den hier relevanten Typen zählen Service (SRV), Text (TXT), Pointer (PTR) und Address (A) Resource Records.

Die Verwendung der Resource Records wird von DNS SD[52] (DNS Service Discovery) näher spezifiziert, um die Suche nach Diensten zu vereinfachen. SRV Resource Records dienen der Beschreibung von Diensten. Das zugrunde liegende Format ist in RFC 1035 beschrieben. In RFC 2782[53] erfolgen Angaben zum RDATA-Bereich. Ein DNS-SD konformer Servicename beinhaltet die Bestandteile Serviceinstanz, Servicetyp, primäres Protokoll und Domain. Hierbei kann die Bezeichnung für die Serviceinstanz frei gewählt werden. Gültige Servicetypen sind über den Link der DNS Organisation abrufbar[54]. Mögliche primäre Protokolle sind *udp* und *tcp*.

Domainnamen im lokalen Kontext sind durch die Pseudo-Topleveldomain *.local*, die mit der Multicast-Adresse 224.0.0.251 assoziiert ist, gekennzeichnet. TXT Resource Records dienen der Parametrisierung von Diensten unter Verwendung von Schlüssel-Werte-Paaren. Diese sind in Form von Zeichenketten, denen ein die Länge kennzeichnendes Byte vorangestellt ist, zu speichern. So ist beispielsweise die Zuordnung einer Versionsnummer zu einem Dienst möglich (0x0B|t|x|t|v|e|r|s|=|1|.|0). Der Einsatz von TXT Resource Records ist optional. Im Vergleich zu SRV Resource Records weisen TXT Resource Records nur im RDATA-Bereich strukturelle Abweichungen auf.

Ein PTR Resource Record stellt eine Referenz auf einen SRV Resource Record dar. Das Element Name beinhaltet Servicetyp, primäres Protokoll und Domain. Das Element PTRDNAME verweist auf die zugehörige Serviceinstanz. Suchanfragen unter Verwendung von PTR Resource Records ermöglichen somit die Selektion einer Menge von Instanzen (z. B. Resource Record 1: PTRDNAME = server1._http._tcp.local; Resource Record 2: PTRDNAME = server2._http.tcp.local) über den Servicenamen (z. B. Name = _http._tcp.local). Die eindeutige Zuordnung von IP-Adressen zu Domainnamen wird auf Grundlage von A Resource Records realisiert. In der Regel existiert zu jedem Target Element eines SRV Resource Records ein entsprechender A Resource Record. Typische Kommunikationsbeziehungen im Zusammenhang mit der Suche nach Dienstattributen mittels mDNS-Anfrage durch einen Client sowie die Bekanntgabe von Dienstattributen mittels mDNS-Mitteilung durch einen Server sind anbei dargestellt. Zunächst wird auf den allgemeinen Ablauf eingegangen. Es folgt die Erläuterung der Funktionsweise anhand eines Lichtschalters, der zum Zwecke der Kommunikationsanalyse entwickelt wurde. Abbildung 22 zeigt die vom Client initiierte Suche nach Dienstattributen mittels mDNS-Anfrage.

---

[49] Rekhter et al.: Address Allocation for Private Internets, RFC 1918.

[50] Chesire/Aboba/Guttman: Dynamic Configuration of IPv4 Link-Local Addresses, RFC 3927.

[51] Mockapetris: Domain Names – Implementation and Specification, RFC 1035.

[52] Chesire/Krochmal: DNS-Based Service Discovery.

[53] Gulbrandsen/Vixie/Esibov: A DNS RR for specifying the location of services (DNS SRV), RFC 2782.

[54] http://www.dns-sd.org, zugegriffen am 05.02.2009.

Zunächst wird ein Zeroconf-kompatibler HTTP-Serverdienst durch *Aktor 1* und im Anschluss mit geringem zeitlichen Versatz ein Zeroconf-kompatibler HTTP-Client durch *Aktor 2* gestartet. Der Client sucht nun nach verfügbaren HTTP-Diensten. Hierzu verschickt er eine mDNS-Anfrage mit entsprechend ausgeprägtem PTR Resource Record. Im Allgemeinen sind mDNS-Anfragen nicht auf die Suche nach Diensten unter Verwendung von PTR Resource Records beschränkt. Grundlage der Suche können auch sonstige Resource Records sein, so z. B. der Address Resource Record im Rahmen der Adressauflösung.

Im Beispiel gibt der Dienst seinen Namen *LightSwitch*, seine Erreichbarkeit über den Port 4242 und seine IP-Adresse 192.168.2.102 mit dem im lokalen Kontext gültigen Hostnamen *compi.local* bekannt. In Kenntnis dieser Informationen kann der Dienst durch *Aktor 2* genutzt werden. Mit dem Bonjour-Plugin als Client-Anwendung ist die zugehörige Benutzerschnittstelle vom Browser aus mit nur einem Mausklick aufrufbar. Statusupdates kommen im Beispiel aus Gründen der Übersichtlichkeit nicht vor. Sie lassen sich mit TXT Resource Records leicht realisieren. Hierzu fragt der Client vom Dienst einen TXT Resource Record an. Diese Anfrage wird dann bei Änderung des Resource Records bedient. Durch Einbettung der Anfrage in eine Endlosschleife kann der Client somit fortlaufend über Statusänderungen informiert werden[55].

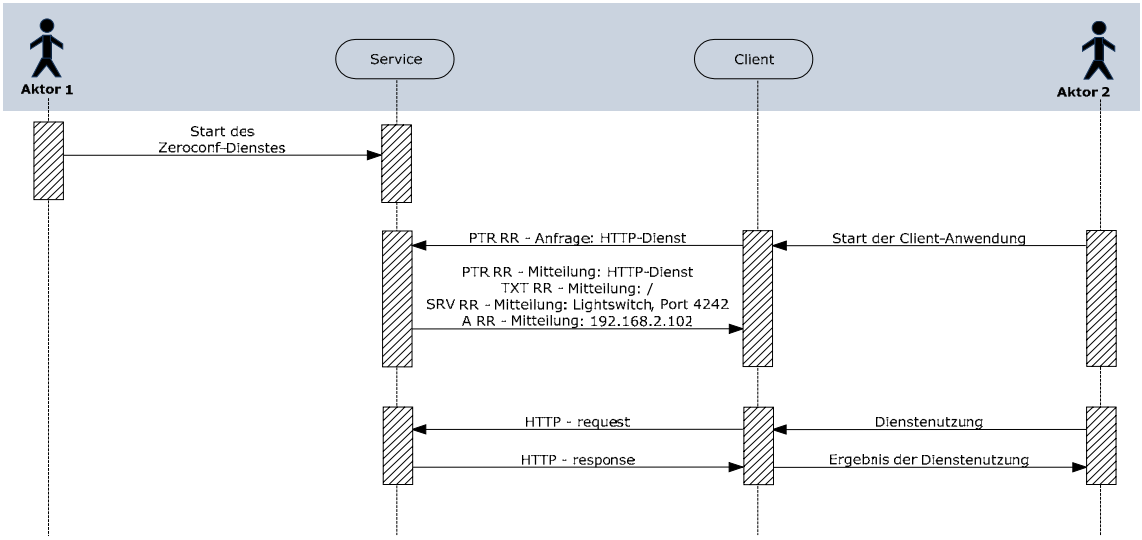


Abbildung 22 Suche nach Dienstattributen durch einen Client. Nachdem der Zeroconf-Dienst durch Aktor 1 gestartet wurde, leitet Aktor 2 die Suche nach HTTP-Diensten ein. Die zurück gelieferten Informationen befähigen zur Nutzung. Suchanfrage und -ergebnis werden über Resource Records (RRs) beschrieben.

Abbildung 23 zeigt die vom Dienst initiierte Bekanntgabe seiner Attribute mittels mDNS-Mitteilung. In diesem Fall wird zunächst ein Zeroconf-kompatibler HTTP-Client durch *Aktor 2* und im Anschluss mit geringem zeitlichen Versatz ein Zeroconf-kompatibler HTTP-Server durch *Aktor 1* gestartet. Unter Verwendung von SRV, TXT, PTR und A Resource Records stellt der Dienst analog dem zuvor genannten Beispiel alle die für seine Nutzung durch *Aktor 2* benötigten Informationen ohne explizite Anfrage bereit. Durch die Unterstützung des Push- und Pull-Mechanismus ist ein Client unabhängig von der Anschlussreihenfolge der Kommunikationsteilnehmer auf einfache Weise über aktuell verfügbare Dienste informiert.

[55] Chesire/Steinberg: Zero Configuration Networking, S. 105 f.

Gleichzeitig wird gegenüber dem reinen Push-Ansatz – gekennzeichnet durch das serverseitige Senden von Dienstbeschreibungen in kurzen Abständen – sowie dem reinen Pull-Ansatz – gekennzeichnet durch die andauernde Suche nach verfügbaren Diensten durch Clients – das Datenaufkommen erheblich reduziert. Ebenso wird keine zentrale Koordination benötigt, da sowohl Dienste-Anfragen als auch -Bekanntgaben Multicast-basiert sind. Hierin liegt ein wesentlicher Unterschied zu Jini mit dem Lookup-Service als koordinierendem Element.

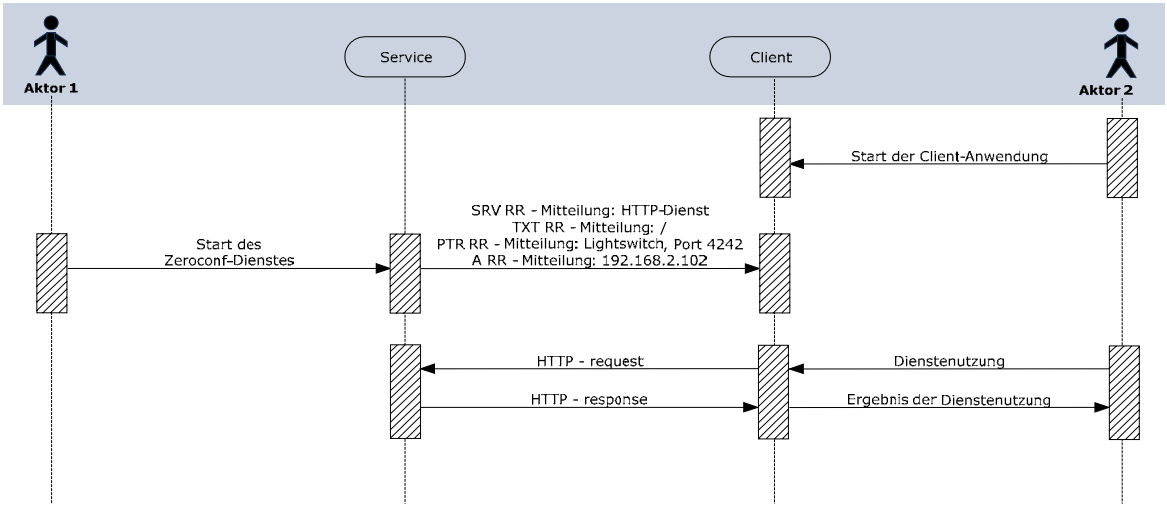


Abbildung 23 Bekanntgabe von Diensten ohne explizite Anfrage. Aktor 2 hat eine Client-Anwendung für HTTP-Dienste bereits gestartet. Im Anschluss hinzukommende Dienste geben die für ihre Nutzung benötigten Attribute unter Verwendung von Resource Records (RRs) unmittelbar bekannt.

Ausgehend von der zuletzt genannten Kommunikationsbeziehung werden wesentliche Protokollelemente vorgestellt. In Tabelle 2 ist die Struktur der mDNS-Beispielnachricht abgebildet. Diese beinhaltet einen Header, eine Antwortsektion sowie eine Zusatzsektion. Die Antwortsektion besteht aus einem SRV, TXT und PTR Resource Record. Die Zusatzsektion umfasst ausschließlich einen A Resource Record. Da es sich bei der mDNS-Beispielnachricht um eine Dienstebekanntgabe handelt, ist die Anfragesektion leer.

Aus Performancegründen wird bei DNS-Antworten die Authoritysektion nur dann verwendet, wenn die an einen DNS-Server gerichtete Anfrage von diesem nicht beantwortet werden kann. Hiermit erhält der Anfragesteller einen Verweis auf zuständige autoritative Server, die weiterführende Informationen bereitstellen[56,57]. Es ist zu beachten, dass die aufgeführten Server autoritativ in Bezug auf die nächste zu kontaktierende Station im Rahmen der Adressauflösung sind. Wird z. B. die iterative Suchanfrage „www.tu-freiberg.de“ an einen DNS-Root-Server gerichtet, wird ein für die Toplevel-Domain „.de“ zuständiger Server – wie „a.nic.de“ – in der Authoritysektion referenziert. Zum Nachvollziehen der Funktionsweise kann beispielsweise das Netzwerktool Dig eingesetzt werden[58]. Bei mDNS-Bekanntgaben ist die Authoritysektion stets leer, offensichtlich beinhaltet die Antwortsektion bereits alle relevanten Informationen für den Dienstzugriff.

[56] Dies kommt im Wesentlichen bei der iterativen Adressauflösung zum Tragen. Bei der rekursiven Adressauflösung hingegen tritt der geschilderte Fall nur dann ein, wenn eine Adressauflösung gänzlich unmöglich ist. Andernfalls wird stets ein Ergebnis zurückgeliefert.

[57] Mockapetris: Domain Names – Implementation and Specification, RFC 1034, S. 43 ff.

[58] Die zum Beispiel gehörige Kommandozeile lautet dann: dig @f.root-servers.net www.tu-freiberg.de +norecurse.

Tabelle 2 Struktur der mDNS-Beispielnachricht. Sie umfasst die drei Bestandteile Header, Antwortsektion und Zusatzsektion.

Nachrichtenstruktur DNS (gemäß RFC 1035)	**Struktur der Beispielnachricht**
Header	mDNS Header
Question section (Anfragesektion)	-
Answer section (Antwortsektion)	SRV Resource Record
	TXT Resource Record
	PTR Resource Record
Authority section (Authoritysektion)	-
Additional section (Zusatzsektion)	A Resource Record

Die Bestandteile der in Tabelle 2 abgebildeten, mit dem Netzwerkprotokollanalyseprogramm Wireshark ermittelten Bytesequenz der mDNS-Beispielnachricht werden exemplarisch einer detaillierten Analyse unterzogen, um charakteristische datenstrukturelle Merkmale herauszustellen. Aus den Abbildungen zum mDNS-Header und den jeweiligen Resource Records geht in Kurzform hervor, welche Bedeutung jedem einzelnen Byte zukommt.

Tabelle 3 Bytesequenz der mDNS-Beispielnachricht. Sie beschreibt den mDNS Header ebenso wie den SRV, TXT, PTR und A Resource Record.

mDNS Header															
00	00	84	00	00	00	00	03	00	00	00	01				
SRV RR															
0b	4c	69	67	68	74	53	77	69	74	63	68	05	5f	68	74
74	70	04	5f	74	63	70	05	6c	6f	63	61	6c	00	00	21
80	01	00	00	00	78	00	0e	00	00	00	00	23	e7	05	63
6f	6d	70	69	c0	23										
TXT RR															
c0	0c	00	10	80	01	00	00	11	94	00	01	00			
PTR RR															
c0	18	00	0c	00	01	00	00	11	94	00	02	c0	0c		
A RR															
c0	3a	00	01	80	01	00	00	00	78	00	04	c0	a8	02	66

Der Beginn einer Zeichenfolge wird mit einem ihre Länge bestimmenden Byte markiert. Für den Abschluss wird ein Nullbyte verwendet. Wiederkehrende Zeichenfolgen werden kodiert, indem die ersten beiden Bits eines Byte-Paares auf 1 gesetzt werden.

Die verbleibenden 14 Bit stellen einen Offset in Bezug auf den Nachrichtenbeginn dar[59]. Das Format des mDNS-Headers ist Abbildung 24 zu entnehmen. Konkrete Datenausprägungen sind durch blaue Schriftfarbe kenntlich gemacht. Nebenstehende Tabelle zeigt die Kurzbeschreibungen der zum mDNS-Header gehörigen Bitfeldgruppen. Für die sich anschließenden Formatbeschreibungen des SRV, TXT, PTR und A Resource Records wird eine analoge Darstellungsform gewählt.

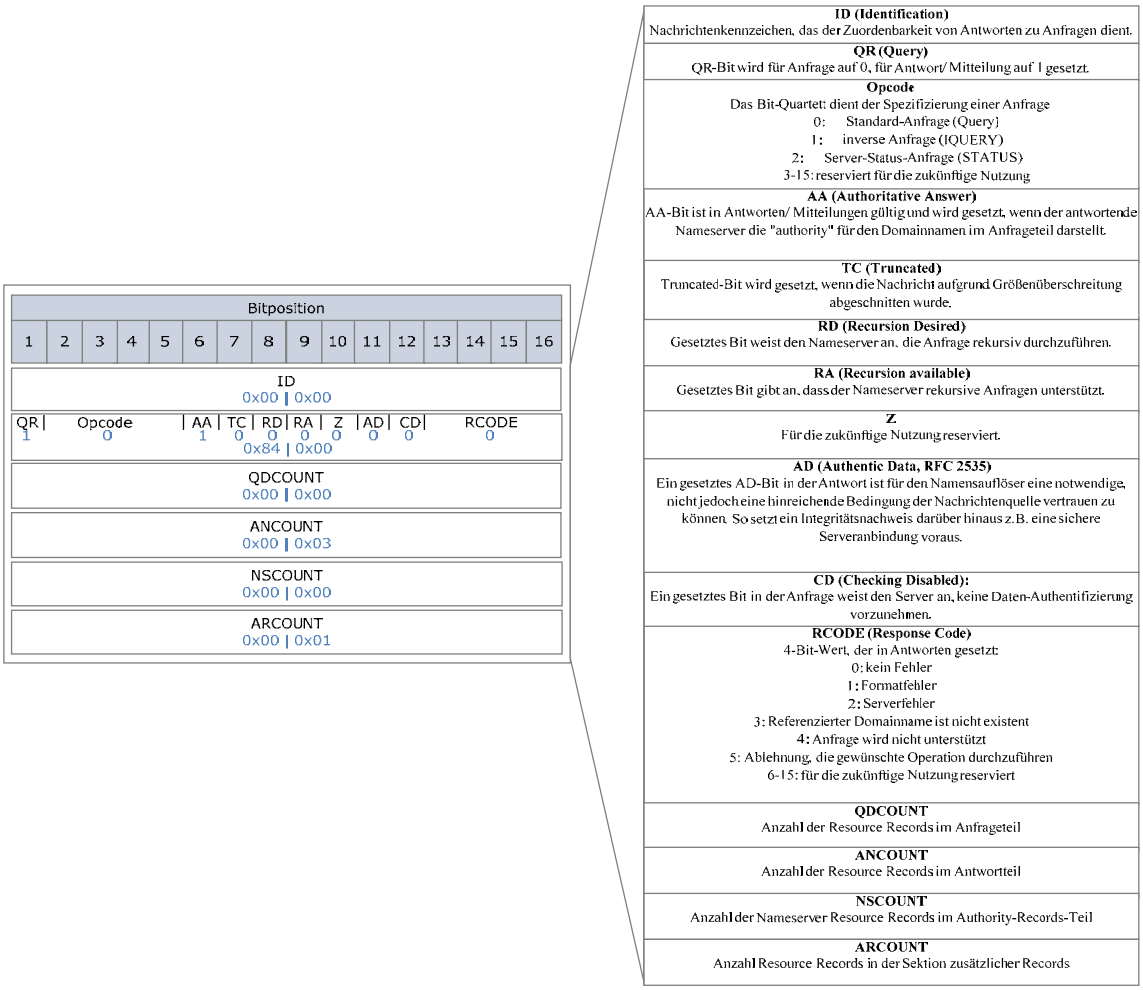


Abbildung 24 Format und Ausprägung des mDNS-Headers[60]. Dieser beschreibt, ob es sich um eine Anfrage- oder Antwortnachricht handelt. Er umfasst darüber hinaus ergänzende Angaben zur Nachricht. Hierzu zählt z. B. die Anzahl der unterschiedlichen Resource Records.

Aus obigen Informationen ist abzulesen, dass es sich um eine Antwort- bzw. Mitteilungsnachricht (QR = 1) mit drei Resource Records im Antwortteil (ANCOUNT = 3) und einem Resource Record im Zusatzteil (ARCOUNT = 1) handelt. Wie bereits in Tabelle 3 angegeben, sind die Anfrage- und Authoritysektionen leer. Dies ist an den aus 0 bestehenden Bitfolgen für QDCOUNT und NSCOUNT zu erkennen. Das AA-Bit wird bei einer DNS-Antwort immer dann gesetzt, wenn sie von einem autoritativen Server kommt. Das ist der für eine Zone zuständige Server. Sofern keine Zwischenspeicherung erfolgt, steht dieser als letztes in der Kaskade der Adressauflösung.

[59] Mockapetris: Domain Names – Implementation and Specification, RFC 1035, S. 30 f.
[60] Mockapetris: Domain Names – Implementation and Specification, RFC 1035, S. 26 ff.

Das AA-Bit wird nicht gesetzt, wenn eine Anfrage aus dem Cache eines für die Zone nicht verantwortlichen Servers bedient wird. Im Internet laufen aus Gründen der Effizienz die meisten Anfragen auf diesen Fall hinaus. Nicht-autoritative Antworten bergen die Gefahr, nicht mehr gültig zu sein. Vorliegend werden die Adressinformationen über den Lightswitch-Dienst von diesem selbst bereitgestellt. Demzufolge ist das AA-Bit in der mDNS-Nachricht gesetzt. Auf die weiteren Datenfelder des DNS-Header wird an dieser Stelle nicht weiter eingegangen, da sie für das Grundverständnis von Zeroconf nicht bedeutsam sind. Analog wird bei den Erläuterungen der unterschiedlichen Resource Records verfahren. Das Format des SRV Resource Records ist Abbildung 25 zu entnehmen. Nebenstehende Tabelle zeigt die Kurzbeschreibungen der zum SRV Resource Record gehörigen Bitfeldgruppen.

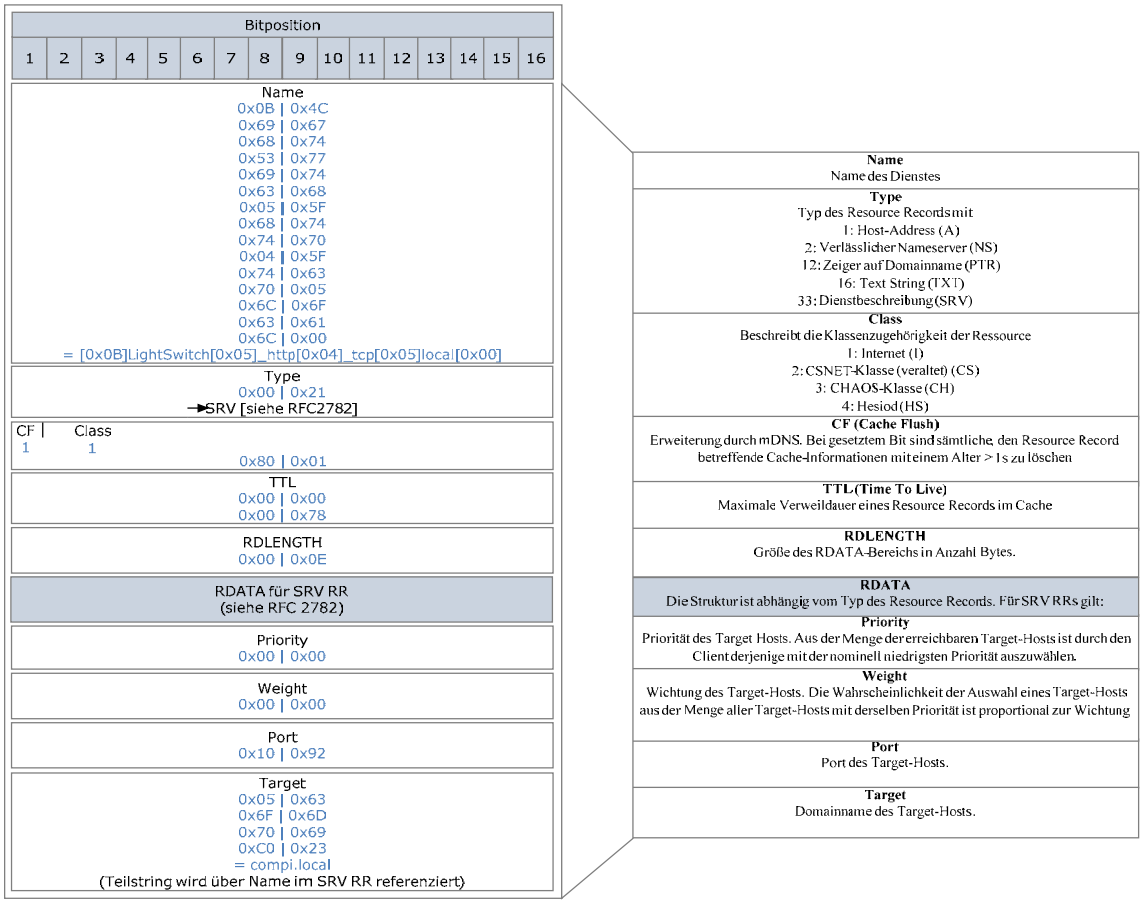


Abbildung 25 Format und Ausprägung des SRV Resource Records (RR)[61,62]. Er dient u. a. der Zuordnung eines Hostnamens und einer Portnummer zu einem Dienst.

Wesentliche aus dem Resource Record extrahierbare Informationen sind der Dienstname (Name = *LightSwitch._http._tcp.local*), der mit dem Dienst assoziierte Hostname (Target = *compi.local*) und die Portnummer (Port = $1092_h = 4242_d$). Mit dem Wert 0x21 = 33 im Type-Feld wird der Resource Record als SRV Resource Record klassifiziert. Weitere Angaben zum Dienst finden sich in dem A Resource Record. Wie bereits erwähnt stimmt der Aufbau des SRV Resource Records grundsätzlich mit dem von TXT, PTR und A Resource Records überein. Strukturell abweichend ist ausschließlich der RDATA-Bereich.

[61] Mockapetris: Domain Names – Implementation and Specification, RFC 1035, S29 f.
[62] Gulbrandsen/Vixie/Esibov: A DNS RR for specifying the location of services (DNS SRV), RFC 2782, S. 2 ff.

Das Format des TXT Resource Records ist Abbildung 26 zu entnehmen. Nebenstehende Tabelle zeigt die Kurzbeschreibungen der zum TXT Resource Record gehörigen Bitfeldgruppen. Die Einstufung als TXT Resource Record geht aus dem Wert 0x10 = 16 im Type-Feld hervor.

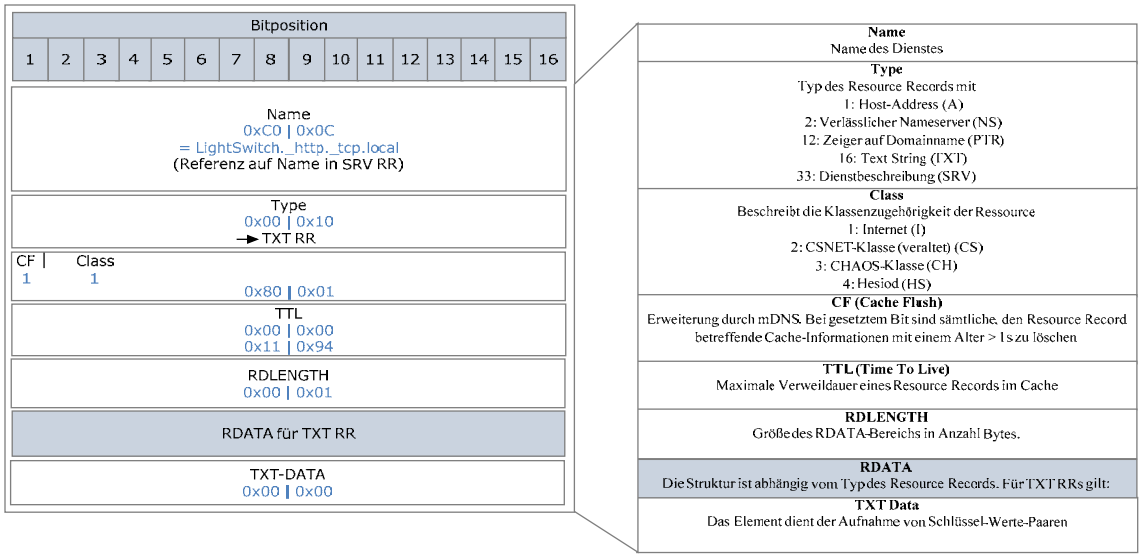


Abbildung 26 Format und Ausprägung des TXT Resource Records (RR). Schlüssel-Werte-Paare werden im Beispiel nicht genutzt, um den Dienst zu qualifizieren. Daher sind alle Bits im TXT DATA-Feld auf 0 gesetzt.

Im Beispiel wird der Dienst *LightSwitch._http._tcp.local* nicht weiter durch Schlüssel-Werte-Paare spezifiziert – alle Bits im TXT DATA-Feld sind auf 0 gesetzt. Das Format des PTR Resource Records ist Abbildung 27 zu entnehmen. Nebenstehende Tabelle zeigt die Kurzbeschreibungen der zum PTR Resource Record gehörigen Bitfeldgruppen.

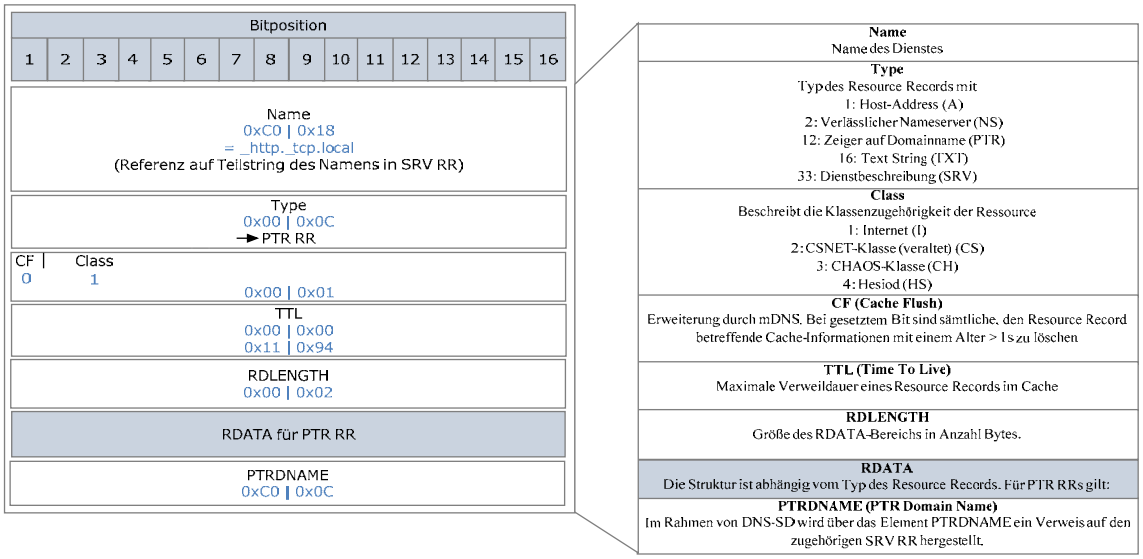


Abbildung 27 Format und Ausprägung des PTR Resource Records (RR)[63]. Im Name-Feld sind der Servicetyp *http*, das primäre Protokoll *tcp* sowie die Domain *local* kodiert. Über das PTRDNAME-Feld wird auf den Dienst *LightSwitch._http._tcp.local* verwiesen. PTR RRs gestatten somit z. B. die Suche nach HTTP-basierten Diensten.

[63] Chesire/Krochmal: DNS-Based Service Discovery, S. 5 ff.

Der Name des Dienstes besteht aus dem Servicetyp *http*, dem primären Protokoll *tcp* und der Domain *local*. Der Wert 0x0C = 12 im Type-Feld ordnet den Resource Record als PTR Resource Record ein. Der PTR Resource Record referenziert über den PTRDNAME-Eintrag den zugehörigen Dienst *LightSwitch* (PTRDNAME = *LightSwitch._http._tcp.local*). Die Suche nach HTTP-basierten Diensten über das PTR-Attribut Name führt demnach zu einer Ergebnismenge, die den Dienst LightSwitch umfasst. Siehe hierzu auch Abbildung 22. Generell werden PTR Resource Records auch für inverses DNS eingesetzt. Die IP-Adresse ist dann im Feld Name kodiert, der RDATA-Bereich ist leer. Die Reihenfolge der einzelnen 8-Bit IP-Blöcke wird umgekehrt und um die Topleveldomain *arpa* sowie Secondleveldomain *in-add* ergänzt[64]. Auf diese Weise werden in Analogie zu DNS-Namen die Angaben von rechts nach links immer spezifischer.

Das Format des A Resource Records ist Abbildung 28 zu entnehmen. Nebenstehende Tabelle zeigt die Kurzbeschreibungen der zum A Resource Record gehörigen Bitfeldgruppen. Die IP-Adresse des Dienste anbietenden Hosts *compi.local* befindet sich im Address-Feld des RDATA-Bereichs. Sie lautet 192.168.2.102.

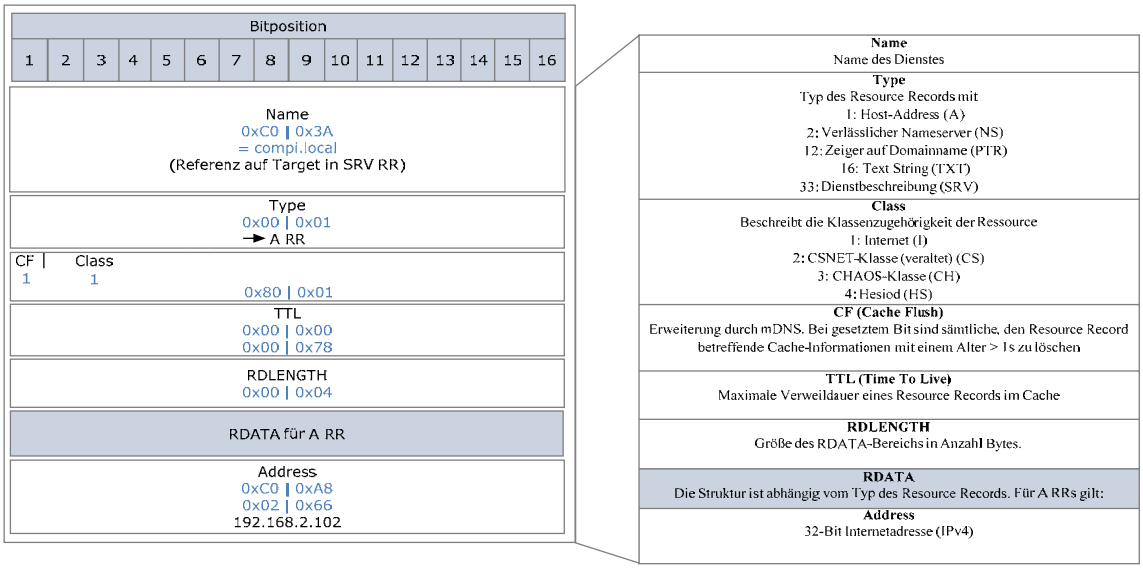


Abbildung 28 Format und Ausprägung des A Resource Records (RR). Aus diesem geht hervor, dass der Host *compi.local* mit der IP-Adresse 192.168.2.102 verknüpft ist.

Hat ein weiterer Dienst D_2 den Namen des verkündenden Dienstes D_1 bereits für sich beansprucht, so ist D_2 in der Pflicht, den detektierten Namenskonflikt mittels Unicast Antwort-Nachricht D_1 mitzuteilen. In der Folge ist durch D_1 solange die Verkündung alternativer Namen auf Grundlage von A Resource Records durchzuführen, bis kein Namenskonflikt mehr signalisiert wird. Der HTTP-Dienst *LightSwitch* ist nach Bekanntgabe durch die beschriebene mDNS-Nachricht in Zeroconf-kompatiblen Browsern (z. B. Safari, Internet Explorer mit Bonjour-Plugin) als Auswahlmöglichkeit aufgeführt. Nach Selektion desselbigen erfolgt die Anzeige des von IP 192.168.2.102, Port 4242 ausgehenden Datenstroms. Im Beispiel ist dies eine HTML-Seite, die es dem Nutzer ermöglicht, eine Lampe in Abhängigkeit ihres aktuellen Zustands über einen Button ein- respektive auszuschalten. Um Dienste in einem WAN zur Verfügung zu stellen, nehmen diese die Registrierung ihrer Resource Records an einem Dynamic DNS Server vor.

[64] Die IP-Adresse der TU Bergakademie Freiberg 139.20.64.26 wird mit 26.64.20.139.in-add.arpa kodiert.

Dieser dient im weiteren Verlauf als zentraler Punkt für mögliche Anfragen durch Clients. Da bei der über das LAN hinausgehenden Kommunikation in der Regel der gesamte private Adressbereich eines Subnetzes auf eine öffentliche IP-Adresse unter Nutzung von NAT[65] abgebildet wird, ist die Zuordnung von im LAN befindlichen Diensten und öffentlicher IP-Adresse $IP_{öff}$ nicht eineindeutig. Um einen im LAN unter der IP IP_{lok} und dem Port $Port_{lok}$ erreichbaren Dienst aus dem öffentlichen Netz referenzieren zu können, bedarf es daher zusätzlich der Angabe eines öffentlichen Ports $Port_{öff}$, der die Assoziation mit dem Dienst ermöglicht.

Diese Form der Abbildung ($IP_{öff}$, $Port_{öff}$) ↔ (IP_{lok}, $Port_{lok}$) wird als Portmapping bezeichnet. Die Administration von Portmapping-Einträgen erfolgt bei Zeroconf-kompatiblen Geräten wahlweise über das NAT-PMP (NAT Port Mapping Protocol) oder das UPnP Home Gateway Protocol. Der Vollständigkeit halber sei an dieser Stelle eine alternative Verfahrensweise erwähnt, mit der trotz NAT von einem Client *C* auf einen Dienst *D* im LAN zugegriffen werden kann. Hierbei fungiert ein im öffentlichen Netzbereich befindlicher Server *S* als Vermittler. *D* baut zunächst eine Verbindung mit *S* auf und teilt eine Identifikationsbezeichnung *I* mit. Damit *C* mit *D* kommunizieren kann, initiiert *C* eine Verbindung mit *S* und spezifiziert den Zielhost unter Angabe der Identifikationsnummer *I*. Im weiteren Kommunikationsverlauf besteht die Aufgabe von *S* in der Weiterleitung von Daten zwischen *C* und *D*. Nach dem beschriebenen Prinzip arbeiten z. B. die Skype-Protokolle[66].

Eine weitere Alternative besteht darin, dass der öffentliche Server ausschließlich während des Verbindungsaufbaus eine vermittelnde Funktion wahrnimmt. Es existiere eine dauerhafte Verbindung von S mit den Teilnehmern T_1 und T_2. Zunächst ermittelt S die öffentlichen IP-Adressen und Ports von T_1 und T_2, indem es von dort ausgehende Datenpakete in Empfang nimmt und die zugehörigen Header auswertet. Gegebenenfalls erfordert das zwischen T_1 und T_2 zu verwendende Protokoll die eigenen externen Adress- und Portangaben. Deren Rücksendung wird u. a. von Session Traversal Utilities for NAT (STUN)[67] geboten. Beispielsweise greift eine mittels Session Initiation Protocol (SIP)[68] initiierte Kommunikation auf diesen Mechanismus zurück, da jedes SIP-Paket stets eine Quelladresse beinhaltet[69]. Das Verbindungsmanagement mit SIP kommt z. B. bei RTP/RTCP[70] zum Einsatz. Die Funktionalität von SIP entspricht der Signalisierung innerhalb verbindungsorientierter Netze wie z. B. ISDN[71].

Damit T_1 eine Verbindung mit T_2 herstellen kann, wird T_2 über S angewiesen, ein Datenpaket an die externe Adresse von T_1 zu senden. Das Paket wird bei T_1 verworfen. Der Sendevorgang führt dazu, dass T_2 im Weiteren Datenpakete von T_1 über den NAT-Router bei T_2 hinweg empfangen kann. T_1 ermittelt im Anschluss über S die externe Adresse von T_2, um die Übertragung von T_1 nach T_2 zu initiieren. Hiermit verbunden ist die Öffnung eines Rückkanals, sodass künftig Datenpakete von T_2 nach T_1 den NAT-Router bei T_1 passieren können. Letztendlich ist somit eine Peer-to-Peer-Kommunikation zwischen T_1 und T_2 möglich. Für die Entwicklung von Zeroconf-Geräten stehen Werkzeuge bereit, z. B. Bonjour SDK für MS Windows oder resolvr für Symbian Smartphones.

---

[65] Srisuresh/Egevang: Traditional IP Network Address Translator (Traditional NAT), RFC 3022.
[66] Kurose/Ross: Computernetzwerke, S. 190 f.
[67] Rosenberg et al.: Session Traversal Utilities for NAT (STUN), RFC 5389.
[68] Rosenberg et al.: SIP: Session Initiation Protocol, RFC 3261.
[69] Der Empfänger eines SIP-Pakets greift direkt auf die darin befindliche Quellangabe zurück und wertet nicht den IP-Header aus, der ebenfalls eine externe Adresse beinhaltet.
[70] Kurose/Ross: Computernetzwerke, S. 674.
[71] Rupp/Siegmund/Lautenschlager: SIP – Multimediale Dienste im Internet, S. 9.

Im Gegensatz zu Jini sind die für die Implementierung der Protokolle erforderlichen Hardware- und Software-Anforderungen gering. Dies gilt sowohl für das zu steuernde als auch das steuernde Gerät.

2.2.3 UPnP

Die Entwicklung von UPnP als Erweiterung von Plug and Play geht auf Bestrebungen der Microsoft Corporation zurück, eine Ad-Hoc-Infrastruktur – analog zu Jini und Zeroconf – bereitzustellen. Die Technologie wurde im Jahr 1999 durch ein Konsortium bestehend aus Herstellern mit hoher Marktdurchdringung eingeführt. Mit Stand 2010 wird UPnP von ca. 900 Herstellern unterstützt[72]. Typische Massenprodukte sind UPnP-fähige Router, Medienrenderer und -server. Kernstück der UPnP-Technologie sind die drei Protokolle SSDP (Simple Service Discovery Protocol), GENA (General Event Notification Architecture) und SOAP (Simple Object Access Protocol). Die Bekanntgabe von Geräten und Diensten sowie die Suche hiernach wird mittels SSDP realisiert, das auf HTTPMU (HTTP Multicast over UDP) und HTTPU (HTTP Unicast over UDP) aufsetzt. GENA wird für Statusupdates verwendet. Als unterlagertes Protokoll fungiert HTTP. Für den Aufruf von Funktionen ist SOAP bestimmt. Siehe hierzu Abbildung 29.

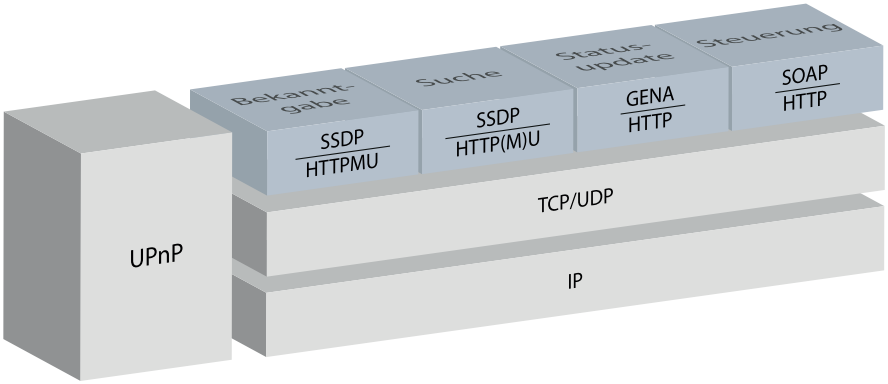


Abbildung 29 Protokollstapel von UPnP. Für die Gerätebekanntgabe und -suche wird das Protokoll SSDP genutzt. Je nach Kontext setzt es entweder auf dem Unicastprotokoll HTTPU oder dem Multicastprotokoll HTTPMU auf. Statusupdates erfolgen mittels GENA. Für die Gerätesteuerung ist das Protokoll SOAP vorgesehen. Sowohl GENA als auch SOAP basieren auf HTTP.

Die Einbindung der im Weiteren beschriebenen Protokollelemente in einen zeitlichen Zusammenhang wird über die in Klammern angegebenen Werte unter Beachtung verschiedener Randbedingungen am Ende des Kapitels vorgenommen (siehe Abbildung 31 und Abbildung 32). Die Bekanntgabe von UPnP-Devices und -Services erfolgt über *ssdp:alive*-Nachrichten. Alle zu diesem Zeitpunkt mit dem Netzwerk verbundenen Clients werden über das zugrunde liegende Protokoll HTTPMU benachrichtigt (1). Ein UPnP-Device kann weitere Devices, die als Embedded Devices bezeichnet werden, beinhalten. Gemäß Spezifikation ist die Verschachtelungstiefe nicht begrenzt.

[72] http://www.upnp.org, aufgerufen am 03.05.2010.

Ist ein UPnP-Device nicht Bestandteil eines anderen, so wird es als Root Device bezeichnet. Der beschriebene Zusammenhang wird in Abbildung 30 dargestellt. Jedes Device stellt eine Menge von Services bereit, wobei die Funktionalität dieser Services über Actions genutzt werden kann. Als synonyme Bezeichnung von Action ist Methode gebräuchlich.

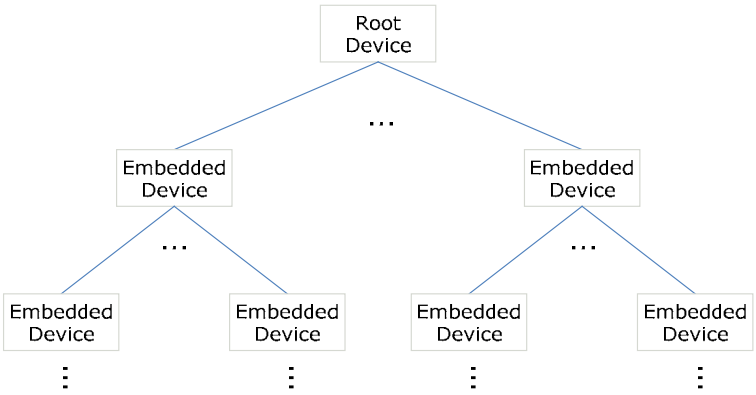


Abbildung 30 Root und Embedded Device im UPnP-Kontext. Jedes Device kann Devices beinhalten. Ist ein Device nicht Bestandteil eines anderen, so wird es als Root Device, andernfalls als Embedded Device bezeichnet.

Eine typische *ssdp:alive*-Nachricht, die mit

NOTIFY * HTTP/1.1

eingeleitet wird, ist nachfolgend dargestellt:

ssdp:alive-Nachricht

```
NOTIFY * HTTP/1.1
NT: upnp:rootdevice
USN: uuid:deb3029d-6301-4e17-acab-e030fb281269::upnp:rootdevice
NTS: ssdp:alive
SERVER: Windows NT/5.0, UPnP/1.0, Intel CLR SDK/1.0
LOCATION: http://127.0.0.1:56361/
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=900
```

Hinsichtlich des HTTP-Requests NOTIFY verweist der SSDP beschreibende Internet Draft[73] im Übrigen auf GENA. Der Typ einer *ssdp:alive*-Nachricht wird über den Header NT (Notification Type) spezifiziert. Entsprechend den Möglichkeiten nach einem Device bzw. Service zu suchen, gibt es für Root Devices drei Ausprägungen, für Embedded Devices zwei Ausprägungen und für Services eine Ausprägung des NT. Die genannten Varianten mit den dazugehörigen Unique Service Names (USNs) für die eindeutige Device- bzw. Servicereferenzierung sind in Tabelle 4 dargestellt.

[73] Goland et al.: Simple Service Discovery Protocol/1.0.

Es sei an dieser Stelle darauf hingewiesen, dass die Bezeichnung USN auch für die Kennung von Devices – und nicht wie der Name zunächst vermuten lässt ausschließlich für Services – verwendet wird.

Tabelle 4 Ausprägungen des NT mit dem dazugehörigen USN.

Device/Service	NT	USN	Beschreibung
Root Device	*uuid:device-UUID*	*uuid:device-UUID*	Eindeutige Geräte-ID
	urn:schemas-upnp-org:device:deviceType:ver	*uuid:device-UUID::urn:schemas-upnp-org:device:deviceType:ver*	Eindeutige Geräte-ID mit Angabe des Gerätetyps und der Versionsnummer
	upnp:rootdevice	*uuid:device-UUID::upnp:rootdevice*	Eindeutige Geräte-ID mit Angabe des Root Device
Embedded Device	*uuid:device-UUID*	*uuid:device-UUID*	Eindeutige Geräte-ID
	urn:schemas-upnp-org:device:deviceType:ver	*uuid:device-UUID::urn:schemas-upnp-org:device:deviceType:ver*	Eindeutige Geräte-ID mit Angabe des Gerätetyps und der Versionsnummer
Service	*urn:schemas-upnp-org:service:serviceType: ver*	*uuid:device-UUID::urn:schemas-upnp-org:service:serviceType:ver*	Servicetyp mit Versionsnummer und der ID des zugehörigen Gerätes

Dem USN ist über die Endung implizit zu entnehmen, über welche Ausprägung des NT er kommuniziert wurde. Somit lassen sich beispielsweise aus einer Liste von USNs all die Einträge selektieren, die ein Root Device repräsentieren, indem die USN-Zeichenkettenendung auf Gleichheit mit „upnp:rootdevice“ geprüft wird. In Summe werden für ein Root Device mit d Embedded Devices und s Services $p = 3 + 2\,d + s$ unterschiedliche *ssdp:alive*-Nachrichten generiert. Die gültigen Header einer *ssdp:alive*-Nachricht sind in Tabelle 5 zusammenfassend aufgeführt[74].

Tabelle 5 Header einer *ssdp:alive*-Nachricht.

Header	Erforderlich	Beschreibung
Host	erforderlich	Angabe des Zielhosts; bei ssdp:alive-Nachrichten ist die UPnP Multicast-Adresse *239.255.255.250:1900* zu verwenden
Cache-Control	erforderlich	Spezifiziert die Gültigkeit einer Notification in Sekunden; muss die Anweisung max-age beinhalten
Location	erforderlich	Diese URL zeigt auf das Dokument der Gerätebeschreibung des Root Device
NT (Notification type)	erforderlich	Nachrichtentyp (siehe Tabelle 4)
NTS (Notification sub type)	erforderlich	Nachrichtensubtyp; muss bei ssdp:alive-Nachrichten *ssdp:alive* lauten
Server	erforderlich	Zusammengesetzte Bezeichnung aus *Betriebssystemname*, *Betriebssystemversion*, UPnP/1.0, *Produktname* und *Produktversion*
USN (Unique Service Name)	erforderlich	Device/Service-Kennung (siehe Tabelle 4)

[74] Jeronimo/Weast: UPnP Design by Example, S. 89.

Ist ein Device oder Service nicht mehr verfügbar, so ist eine *ssdp:byebye*-Nachricht zu generieren (8). Die auf das Root Device der *ssdp:alive*-Nachricht Bezug nehmende *ssdp:byebye*-Nachricht lautet:

ssdp:byebye-Nachricht

```
NOTIFY * HTTP/1.1
NT: upnp:rootdevice
USN: uuid:deb3029d-6301-4e17-acab-e030fb281269::upnp:rootdevice
NTS: ssdp:byebye
HOST: 239.255.255.250:1900
```

Es ist ersichtlich, dass *ssdp:byebye*-Nachrichten genau wie *ssdp:alive*-Nachrichten mit dem HTTP-Request NOTIFY eingeleitet werden. Die gültigen Header einer *ssdp:byebye*-Nachricht sind in Tabelle 6 zusammenfassend aufgeführt[75].

Tabelle 6 Header einer *ssdp:byebye*-Nachricht.

Header	Erforderlich	Beschreibung
Host	erforderlich	Angabe des Zielhosts; bei *ssdp:byebye*-Nachrichten ist die UPnP Multicast-Adresse *239.255.255.250:1900* zu verwenden
NT (Notification type)	erforderlich	Entspricht der Angabe des NT in der *ssdp:alive*-Nachricht
NTS (Notification sub type)	erforderlich	Nachrichtensubtyp; muss bei ssdp:byebye-Nachrichten *ssdp:bye-bye* lauten
USN (Unique service name)	erforderlich	Entspricht der Angabe des USN in der *ssdp:alive*-Nachricht

Mittels Multicast *ssdp:discover*-Nachrichten, die ebenfalls auf dem Protokoll HTTPMU basieren, können Clients nach Devices und Services suchen (2). Der zugehörige HTTP-Request lautet M-SEARCH.

ssdp:discover-Nachricht

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
Man: ssdp:discover
MX: 5
ST: upnp:rootdevice
```

[75] Jeronimo/Weast: UPnP Design by Example, S. 91.

Die gültigen Header einer *ssdp:discover*-Nachricht sind in Tabelle 7 zusammenfassend aufgeführt[76]. Das auf NOTIFY und M-SEARCH folgende Symbol * gibt an, dass die jeweilige Anfrage an alle im Netzwerk befindlichen UPnP-Geräte gerichtet ist.

Tabelle 7 Header einer *ssdp:discover*-Nachricht.

Header	Erforderlich	Beschreibung
Host	erforderlich	Angabe des Zielhosts; bei *ssdp:discover*-Nachrichten ist die UPnP Multicast-Adresse *239.255.255.250:1900* zu verwenden
Man[77] (Mandatory extension)	erforderlich	Muss bei *ssdp:discover*-Nachrichten *ssdp:discover* lauten
MX (Maximum)	erforderlich	Maximale Dauer in Sekunden, nach der eine Antwort vorliegen muss.
ST (Search target)	erforderlich	Dient der Angabe des gesuchten Device/Service; kann die Werte der NT-Ausprägungen aus Tabelle 4 annehmen sowie *ssdp:all* für die Suche nach allen verfügbaren Devices

Genügt ein Device oder ein mit ihm verbundener Service den Suchkriterien, sendet das Device eine auf HTTPU basierende Unicast *ssdp:alive*-Antwortnachricht (3). Das GENA-Protokoll ermöglicht es Clients, sich für asynchrone Ereignisse, die bei Änderung von Zustandsvariablen ausgelöst werden, zu registrieren (4). Bei Eintritt eines Ereignisses werden den hierfür registrierten Clients die aktuellen Werte der Zustandsvariablen mitgeteilt (5). Durch den Push-Mechanismus wird sichergestellt, dass die Änderungen von Zustandsvariablen unmittelbar an den Control Point kommuniziert werden bei gleichzeitiger Minimierung der Netzwerklast. Würde hingegen der Pull-Mechanismus zum Einsatz kommen, müsste in bestimmten Zyklen die Abfrage der zu beobachtenden Zustandsvariablen durch den Control Point erfolgen, unabhängig davon, ob tatsächlich Änderungen vollzogen wurden oder nicht.

UPnP-konforme Prozedurfernaufrufe (Remote Procedure Calls, RPC) basieren auf dem Simple Object Access Protocol, bei dem die übertragenen Nachrichten im XML-Format kodiert werden. Alternativ können die beteiligten Devices für die Steuerung beliebige Protokolle vereinbaren (6). Die beschriebenen Schritte werden am Beispiel von zwei Szenarien visualisiert und in einen zeitlichen Zusammenhang gebracht. Hierbei wird die fallweise Relevanz der Schritte 1 bis 3 berücksichtigt.

Bei dem in Abbildung 31 dargestellten beispielhaften Kommunikationsprozess ist der Start des Control Point dem Anschluss des UPnP-Device zeitlich vorgelagert. In diesem Szenario 1 erfolgt eine Multicast-basierte Bekanntgabe der Devices/Services des UPnP-Device *U1* ohne vorherige Suchanfrage des Control Point (1). Es schließt sich die Registrierung für asynchrone Ereignisse an (4). Im nächsten Schritt wird der Control Point über ein eingetretenes Ereignis informiert (5). Der dann folgende Aufruf einer Gerätefunktion führt zur Rückgabe des zugehörigen Ergebnisses (6–7). Schlussendlich wird das Entfernen des Gerätes von einer *ssdp:byebye*-Nachricht begleitet (8).

[76] Jeronimo/Weast: UPnP Design by Example, S. 82.

[77] Der Header Man ist Teil des „HTTP extension framework“. Der Empfänger muss die Anfrage interpretieren können oder den Statuscode „510 (Not extended)“ zurückgeben.

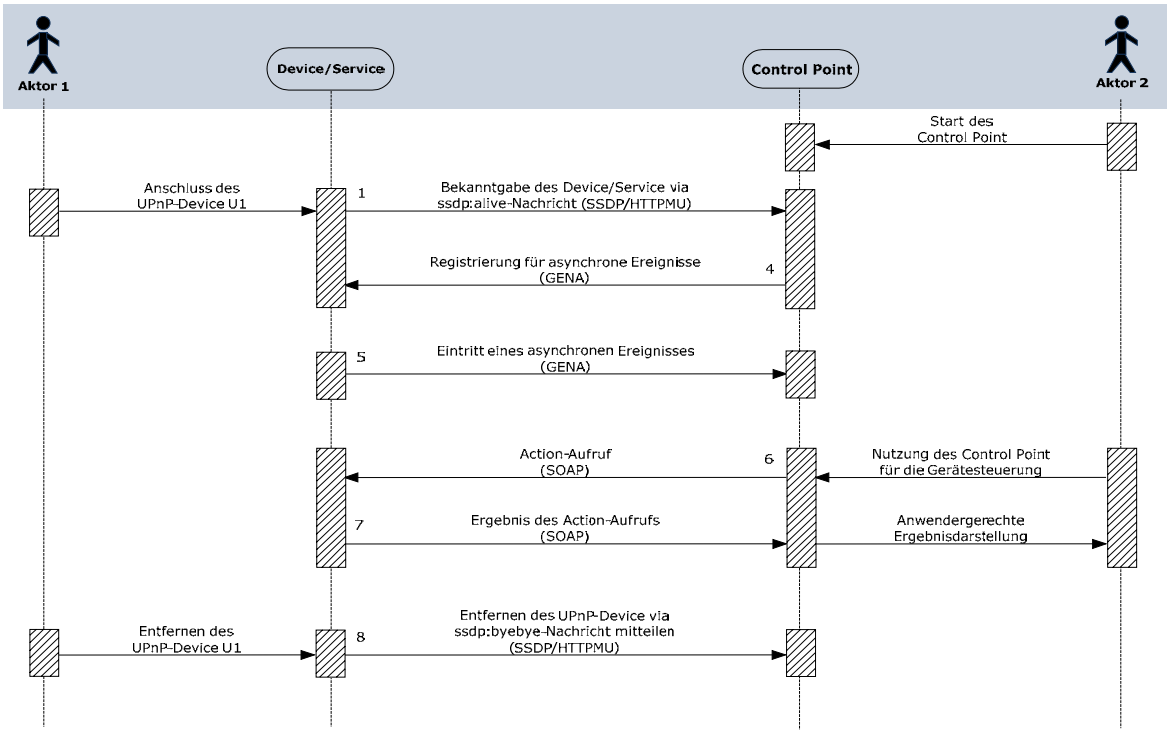


Abbildung 31 Beispielhafte Kommunikation im UPnP-Verbund (Szenario 1). Der von *Aktor 2* bereits gestartete Control Point empfängt eine Präsenzmeldung des UPnP-Device *U1*. Im Weiteren wird die Funktionalität der General Event Notification Architecture genutzt. Es folgt die Steuerung von *U1* durch den Control Point. Abschließend entfernt *Aktor 1* das Gerät.

Bei dem in Abbildung 32 dargestellten beispielhaften Kommunikationsprozess ist der Start des Control Point dem Anschluss des UPnP-Device zeitlich nachgelagert. In Szenario 2 wird eine explizit durchgeführte, Multicast-basierte Suchanfrage des Control Point von den verfügbaren Geräten mit ihren Diensten beantwortet (2–3). Der weitere Verlauf ist identisch mit Szenario 2. Es folgt die Registrierung für asynchrone Ereignisse (4). Kurze Zeit später tritt ein Ereignis ein, das kommuniziert wird (5). Im Rahmen der Gerätesteuerung durch *Aktor 2* initiiert der Control Point einen Funktionsaufruf, dessen Rückgabewert er umgehend erhält (6–7). Den Abschluss bildet die mit dem Entfernen des Gerätes einhergehende *ssdp:byebye*-Nachricht (8).

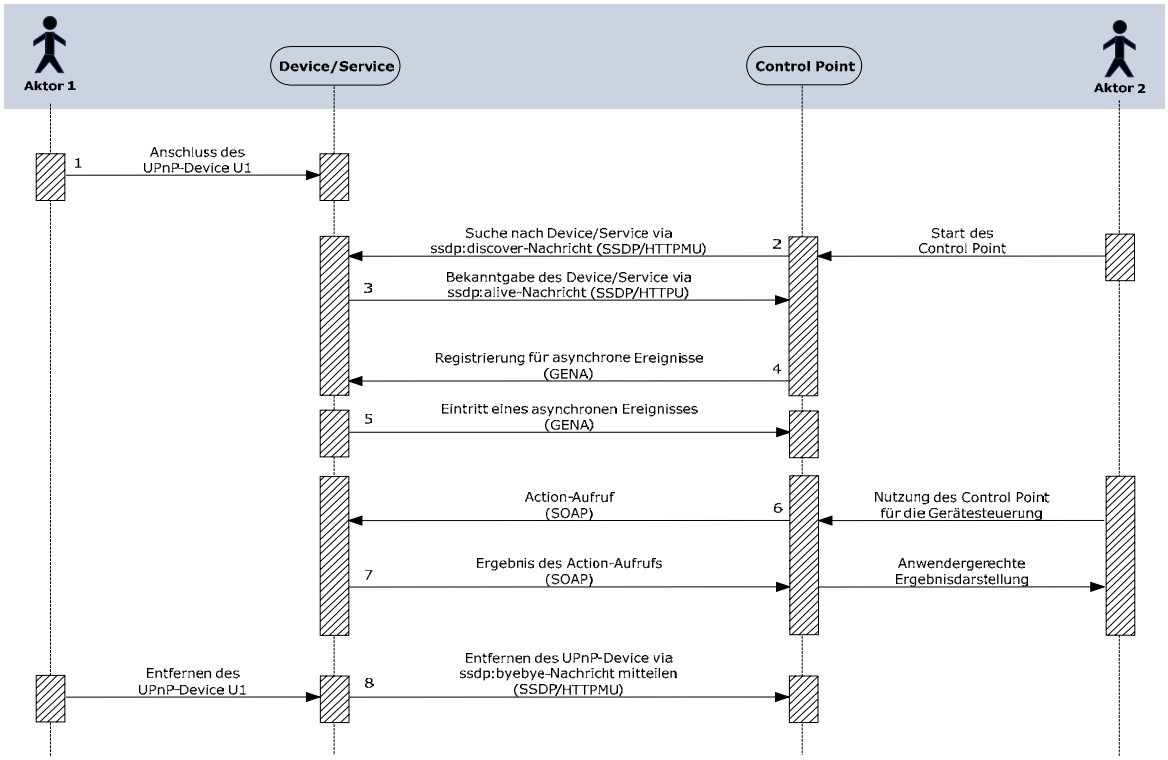


Abbildung 32 Beispielhafte Kommunikation im UPnP-Verbund (Szenario 2). Der Control Point wird kurze Zeit nach dem Anschluss des UPnP-Device *U1* gestartet. Hiermit wird dann nach verfügbaren Geräten gesucht. Die sich anschließenden Schritte von Bekanntgabe bis zum Entfernen des Gerätes *U1* stimmen mit Szenario 1 überein.

Entwicklungskits, die den UPnP Protokollstapel implementieren, sind für zahlreiche Betriebssysteme wie Windows XP, Windows CE und Programmiersprachen wie C++, C# erhältlich. Hardware- und Softwareanforderungen sind wie bei Zeroconf niedrig. Analog zu der Jini-Netzwerkoption existieren auch bei UPnP Ansätze, den funktionsreichen Protokollstapel auszulagern und durch einen minimalistischen zu ersetzen. Zu nennen sei beispielsweise die Sindrion Architektur. Die Implementierung umfasst Plugins mit denen die Integration in das Eclipse Framework möglich ist. Für die Kommunikation zwischen Embedded Devices und dem Sindrion Terminal kommt ein sehr einfach zu implementierendes binäres Protokoll zum Einsatz. Nach außen kommuniziert das als Mittler fungierende Terminal unter Verwendung des UPnP-Protokollstapels[78]. Für die Gerätesteuerung ist ein herkömmlicher UPnP Control Point zu verwenden. Hiervon ausgehend sind low-cost Embedded Devices somit transparent im Zugriff. Das Architekturkonzept gestattet gleichermaßen, weiterhin konventionelle UPnP-Geräte zu integrieren. In der Abbildung ist dies durch die Einbindung einer UPnP-fähigen Lampe kenntlich gemacht. Siehe hierzu Abbildung 33.

[78] Budde/Barisic: Smart UPnP Transceivers, S. 9 ff.

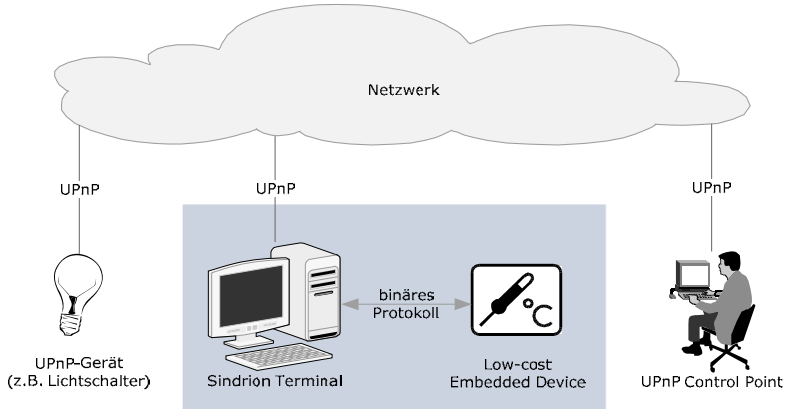


Abbildung 33 Sindrion Architektur. Low-cost Embedded Devices verwenden ein einfach zu implementierendes Protokoll für den Austausch mit einem Terminal. Dieses übernimmt die UPnP-basierte Kommunikation.

Vor dem Hintergrund stetig steigender Leistungsfähigkeit mikroelektronischer Bauelemente wie Speicher und CPUs bei gleichzeitig fallenden Preisen wird auf die Details der Auslagerung des UPnP-Protokollstapels nicht weiter eingegangen. Da die UPnP-Technologie Bestandteil gängiger Microsoft Betriebssysteme ist, erschließt sich ein großer potentieller Nutzerkreis. Darüber hinaus setzen zahlreiche Hardware-Hersteller in ihren Produkten auf die UPnP-Technologie. So wird z. B. das UPnP Home Gateway Protocol[79] durch den Großteil herkömmlicher Router unterstützt.

2.2.4 IEEE 1451

Langjährige Vereinheitlichungsbestrebungen in der Mess-, Steuer- und Regelungstechnik haben zur Entwicklung der IEEE 1451-Standardfamilie geführt. Die Definition der zugrunde liegenden Kommunikations- und Hardwareschnittstellen erfolgte mit der Zielstellung, eine hersteller- und netzwerkunabhängige Gesamtlösung zu etablieren. Es wird grundsätzlich zwischen Sensoren und Aktoren, die übergreifend auch als Transduktoren bezeichnet werden, unterschieden.

Die zentrale Grundidee der Autoidentifikation- und -beschreibung wird mittels sogenannter Transducer Electronic Data Sheets (TEDS) realisiert. Diese ermöglichen die Abkehr von potentiell fehlerbehafteten, manuellen Dateneingaben. Die grundlegenden Ansätze der IEEE 1451-Standardfamilie gehen aus den Teilen 1451.0–1451.6 hervor. Nachfolgend werden deren wechselseitige Beziehungen dargestellt. Abbildung 34 gibt einen Überblick.

[79] Iyer/Warrier: InternetGatewayDevice:1 – Device Template Version 1.01.

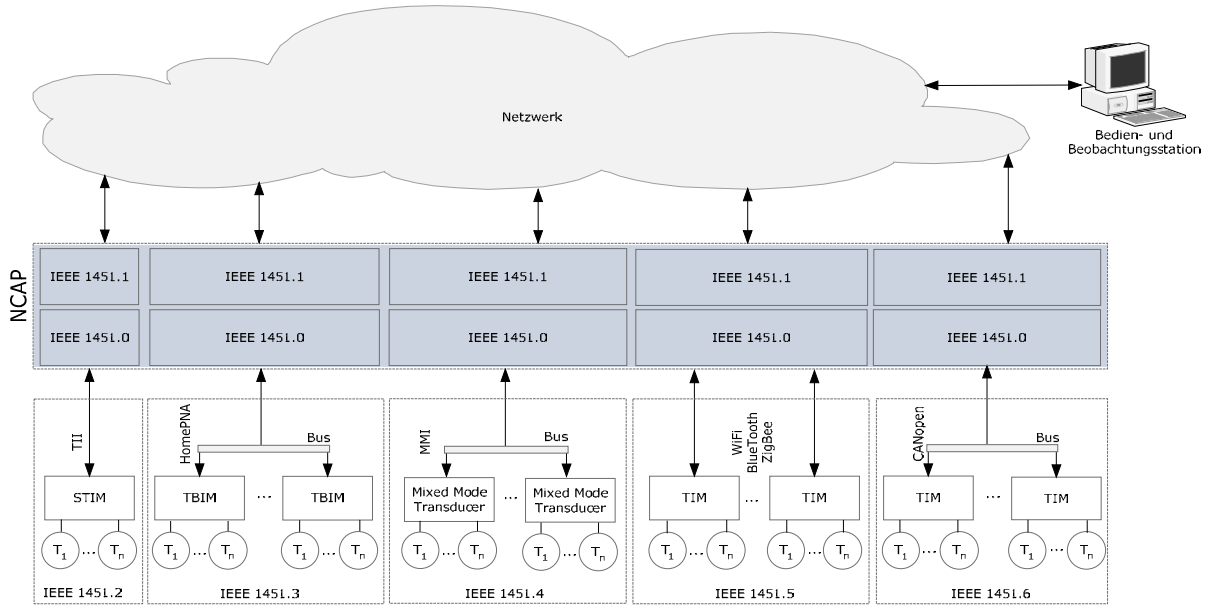


Abbildung 34 Die IEEE 1451-Standardfamilie in einer vernetzten Umgebung. IEEE 1451.2–1451.6 beschreiben Transduktorvarianten. Deren Integration in eine Netzwerkumgebung erfolgt unabhängig von den physikalischen Eigenschaften über den Network Capable Application Processor.

Den physikalischen Transduktorvarianten nach IEEE 1451.2–1451.6 ist gemein, dass das Transduktormanagement wie z. B. das Auslesen und Setzen von Werten sowie die Datenweiterverarbeitung in einer vernetzen Umgebung durch das Network Capable Application Processor (NCAP) erfolgt. Integraler Bestandteil des NCAP sind die Implementierungen der Standards IEEE 1451.0 und IEEE 1451.1. In einer IEEE 1451.4-Umgebung kann an die Stelle des NCAP alternativ ein lokal operierendes Datenerfassungssystem rücken wie z. B. ein System zur Archivierung und Visualisierung von Messwerten. Im Folgenden werden IEEE 1451.0–1451.6 näher vorgestellt. Der Standard IEEE 1451.0[80] stellt die Basis für die Interoperabilität der IEEE 1451-Familie dar. Unabhängig von den zur Anwendung kommenden physikalischen Medien nach IEEE 1451.2–1451.6 mit deren jeweiligen Spezifika wird mit IEEE 1451.0 die Basisfunktionalität für das Zusammenwirken zwischen NCAP und Transduktoren dargeboten. Die Basisfunktionalität umfasst medienunabhängige Funktionen, Kommunikationsprotokolle und TEDS.

Der netzwerkbasierte Zugriff auf das Systemmodell bestehend aus einem Transduktor respektive einer Gruppe von Transduktoren ist Gegenstand des Standards IEEE 1451.1[81]. Das ganzheitliche Modell auf NCAP-Applikationsebene gliedert sich in Objekt-, Daten- und Netzwerkkommunikationsmodell. Das Objektmodell bildet die Funktionalität des Netzwerkgerätes auf eine objektorientierte Darstellungsform ab. Die Beschreibung der zum Einsatz kommenden Datentypen erfolgt im Datenmodell. Hierin wird die Informationskodierung für den Datenaustausch festgelegt. Wesentliches Merkmal des Netzwerkkommunikationsmodells ist die Kapselung von netzwerkspezifischen Charakteristika. Es werden sowohl Client-Server- als auch Publisher-Subscriber-basierte Kommunikationsformen unterstützt.

[80] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats.
[81] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model.

IEEE 1451.1 beinhaltet die Definition zahlreicher Standardobjekte, mit denen Objekt-, Daten- und Netzwerkkommunikationsmodell formalisiert umzusetzen sind. Hierzu zählen beispielsweise Objekte für die Realisierung der netzwerkunabhängigen Kommunikation zwischen NCAP und einer Bedien- und Beobachtungsstation (BuB) sowie Objekte, auf dessen Grundlage der Zugriff der NCAP-Applikation auf die Transduktorvarianten nach 1451.2–1451.6 erfolgt. Die Einbindung der mit den Standards IEEE 1451.0–1451.6 assoziierten Protokollstapel in den Gesamtkontext der Kommunikation zwischen einer BuB sowie einem Transduktor sind in Abbildung 35 dargestellt.

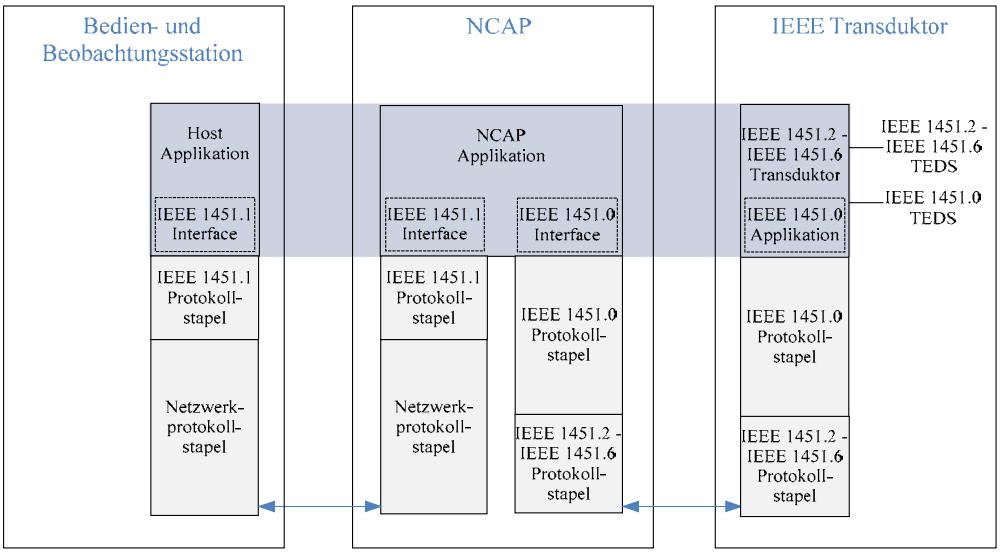


Abbildung 35 Kommunikation zwischen einer Bedien- und Beobachtungsstation sowie einem Transduktor. Der Network Capable Application Processor fungiert als Zwischenstation. Host-seitig bildet IEEE 1451.1, Transduktor-seitig IEEE 1451.0 die Grundlage.

Der Standard IEEE 1451.2[82] definiert ein Smart Transducer Interface Module (STIM), das bis zu 255 einzelne Transduktoren beinhalten kann. Diese Transduktoren werden über die im STIM befindlichen TEDS beschrieben. Im Abschnitt zu TEDS wird auf den Aufbau der IEEE 1451.2 TEDS näher eingegangen. Auf vertiefende Ausführungen zu den TEDS der Standards 1451.3–1451.6 wird aufgrund des konzeptuell analogen Ansatzes verzichtet. Die Anbindung des STIM an das NCAP erfolgt über das Transducer Independent Interface (TII). Über das TTI werden ausschließlich digitale Daten übertragen. Im Rahmen der aktuellen Überarbeitung des Standards P1451.2 (P = Preliminary) durch die IEEE 1451.2 Arbeitsgruppe wird die zusätzliche Unterstützung der weit verbreiteten Schnittstellentechnologien Recommended Standard 232 (RS232), Universal Asynchronous Receiver/Transmitter (UART), Serial Parallel Interface (SPI) und Universal Serial Bus in Erwägung gezogen[83]. Mit dem IEEE 1451.3[84] Standard wird das Bussystem HomePNA (Home Phoneline Networking Alliance) definiert, an das bis zu 255 Transducer Bus Interface Modules (TBIMs) angeschlossen werden können.

[82] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheets (TEDS) formats.
[83] Song/Lee: Sensor Network based on IEEE 1451.0 and IEEE p1451.2-RS232.
[84] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems.

Ein TBIM selbst kann wiederum bis zu 255 Transduktoren beinhalten. Insgesamt besteht also die Möglichkeit, 65.025 Transduktoren über ein NCAP mit HomePNA-Schnittstelle zu adressieren. Die beschriebene Netzstruktur mit der Eigenschaft der Mehrpunktanschlussfähigkeit erlaubt die räumliche Trennung von Transduktoren. Im Englischen wird diese Struktur auch als Distributed Multidrop System bezeichnet. Die Beschreibung der Transduktoreigenschaften erfolgt über die im TBIM befindlichen TEDS. Den Übertragungsmedien HomePNA und TTI ist gemein, dass ausschließlich digitale Daten übertragen werden.

Der digitalen Kommunikation sind die – teilweise erforderlichen, teilweise fakultativen – Schritte analoge Signalkonditionierung (z. B. Signalverstärkung), Analog-Digital-Konvertierung und digitale Signalverarbeitung (z. B. Filtern) vorgelagert. Diese werden bei den Transduktorvarianten nach IEEE 1451.2 und IEEE 1451.3 durch die Transduktor-STIM- bzw. Transduktor-TBIM-Einheit realisiert. Nach IEEE 1451.4[85] wird diese Funktionalität über ein NCAP oder ein lokal operierendes Datenerfassungssystem bereitgestellt, da IEEE 1451.4 konforme Transduktoren im sogenannten Mixed Mode arbeiten. Hierbei werden TEDS-Daten digital und Signaldaten analog übertragen. Aufgrund des resultierenden – gegenüber den Transduktorvarianten nach IEEE 1451.2, IEEE 1451.3, IEEE 1451.5 und IEEE 1451.6 – relativ niedrigen Fertigungsaufwands sind IEEE 1451.4 überwiegend im Niedrigpreissegment angesiedelt.

IEEE 1451.4 unterstützt sowohl die sequentielle Übertragung von TEDS und Signal auf denselben Kabelsträngen (Class 1 Modus) als auch die parallele Übertragung von TEDS und Signal auf unterschiedlichen Kabelsträngen (Class 2 Modus). Der Anschluss eines IEEE 1451.4 Transduktors oder einer Gruppe von IEEE 1451.4 Transduktoren erfolgt über das Mixed Mode Interface (MMI) [86]. Jeder IEEE 1451.4 Transduktor kann wiederum mehrere Transduktoren beinhalten. Die zugehörigen TEDS befinden sich in separaten Speichern, die auch als Nodes bezeichnet werden. Kennzeichen der IEEE 1451.4 TEDS ist der sehr geringe Speicherplatzbedarf. Abbildung 36 zeigt einen IEEE 1451.4 konformen Beschleunigungssensor[87].

Abbildung 36 IEEE 1451.4 Transduktor. Der Beschleunigungssensor integriert Transducer Electronic Data Sheets, die alle für den Betrieb benötigten Informationen bereithalten.

Gegenstand des Standards IEEE 1451.5[88] ist die Definition von Schnittstellen für die drahtungebundene Kommunikation zwischen IEEE 1451.5 Transduktoren und dem NCAP. Es werden die Kommunikationsprotokolle IEEE 802.11a/b/g (WiFi), IEEE 802.15.1 (Bluetooth) und IEEE 802.15.4 (ZigBee) unterstützt.

---

[85] IEEE Standard for A Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Format.

[86] Lee: Smart Transducer Interface Standards for Condition Monitoring and Control of Machines.

[87] http://www.endevco.com, aufgerufen am 01.06.2010.

[88] IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats.

Die Kommunikation mit dem NCAP findet über das Transducer Interface Module (TIM) statt. Die darin befindlichen TEDS dienen der Beschreibung von bis zu 255 mit dem TIM verbundenen Transduktoren. Das in der Automatisierungstechnik weit verbreitete Hochgeschwindigkeitsbussystem CANopen ist Gegenstand des Standards IEEE 1451.6[89]. Es dient der Kommunikation zwischen NCAP und TIM. Das TIM beinhaltet TEDS und ermöglicht den Anschluss von bis zu 255 Transduktoren. Insbesondere im Hinblick auf die Netztopologien und funktionalen Ausprägungen des TIM liegen den Standards IEEE 1451.3 und IEEE 1451.6 identische Ansätze zugrunde. CANopen-basierte Systemausführungen, die den Anspruch der inhärenten Sicherheit (Intrinsic Safety) erfüllen, eignen sich im Besonderen für den Einsatz in Gefahrenbereichen (z. B. Explosionsschutzzonen). Inhärent sichere Systeme zeichnen sich dadurch aus, dass im ordnungsgemäßen Zustand Funkenbildung ausgeschlossen ist und die beinhalteten Bauteilgruppen auch bei Systemfehlverhalten vorgegebene Temperaturgrenzen nicht überschreiten.

Am Beispiel des Standards IEEE 1451.2 wird nachfolgend die Struktur von TEDS Datenblöcken vorgestellt. Diese beinhalten logisch zusammenhängende, transduktorspezifische Informationen. Die Standards IEEE 1451.3–1451.6 folgen – wie bereits beschrieben – konzeptuell ähnlichen Ansätzen, sodass die hierin definierten TEDS zum Standard IEEE 1451.2 analoge Strukturen aufweisen. In IEEE 1451.2 wird zwischen folgenden Datenblöcken unterschieden:

- Meta-TEDS Datenblock
- Channel TEDS Datenblock
- Calibration TEDS Datenblock
- Meta-Identification TEDS Datenblock
- Channel Identification TEDS Datenblock
- Calibration Identification TEDS Datenblock
- End-Users' Application-Specific TEDS Datenblock
- Generic Extensions TEDS Datenblock

Die Datenfelder der beiden zuerst genannten Datenblöcke Meta-TEDS und Channel TEDS sind obligatorisch zu füllen. Die Nutzung der sechs übrigen Datenblöcke ist optional. Bei diesen Datenblöcken kann daher das betreffende „Length"-Feld den Wert 0 annehmen. Im Folgenden werden die acht verschiedenen Typen von Datenblöcken vorgestellt.

Die Informationen des Meta-TEDS Datenblocks beziehen sich auf das gesamte Transduktormodul (STIM), d. h. auf die Gesamtheit aller assoziierten Channels (= Datenkanäle). Wesentliche, für das Grundverständnis relevante, Inhalte der Subblöcke Versionsbezogener Subblock, Identifikationsbezogener Subblock, Datenstrukturbezogener Subblock, Zeitverhaltenbezogener Subblock, Gruppierungsbezogener Subblock und Datenintegritätsbezogener Subblock sind zusammenfassend in Tabelle 8 dargestellt. Auf eine explizite Nennung aller Datenfelder wird aus Gründen der Übersichtlichkeit an dieser Stelle verzichtet. Bedarfsweise ist der Standard IEEE 1451.2 heranzuziehen. Dies gilt für die übrigen TEDS Datenblöcke in gleichem Maße.

[89] IEEE Standard for A Smart Transducer Interface for Sensors and Actuators – A High-speed CANopen-based Transducer Network Interface for Intrinsically Safe and Non-intrinsically Safe Applications.

Tabelle 8 Struktur des Meta-TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Versionsbezogener Subblock	Dieser Subblock beinhaltet Angaben zur Länge des Meta-TEDS, zum genutzten IEEE 1451 Standard (hier: 1451.2) sowie zum aktuellen Versionsstand.
4	Identifikations-bezogener Subblock	Dieser Subblock beinhaltet eine eindeutige Identifikationsnummer für das STIM.
5 bis 12	Datenstruktur-bezogener Subblock	In diesem Subblock erfolgen u. a. Angaben zu gegebenenfalls vorhandenen TEDS-Erweiterungen und zur Anzahl implementierter Channels.
13 bis 23	Zeitverhalten-bezogener Subblock	Angaben, die das Zeitverhalten des STIM charakterisieren, sind in diesem Subblock zusammengefasst. Beispielhaft zu nennen sei das Datenfeld „Command Response Time“, mit dem die maximale Befehlverarbeitungsdauer in Sekunden spezifiziert wird.
24 bis 28	Gruppierungs-bezogener Subblock	Mit den Datenfeldern diese Subblocks können zusammengehörige Channels in eine semantische Beziehung zueinander gesetzt werden. Hierfür stehen verschiedene Typen von Relationen zur Verfügung. Hiermit kann z. B. festgelegt werden, dass die Daten der Channels A, B, C in dieser Reihenfolge als x-, y- und z-Werte (oder etwa als R-, G-, B-Farbwerte) zu interpretieren sind.
29	Datenintegritäts-bezogener Subblock	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Im Gegensatz zu dem Meta-TEDS Datenblock bezieht sich der Channel TEDS Datenblock nicht auf das gesamte Transduktormodul, sondern auf einen der bis zu 255 im STIM integrierten Channels. Der strukturelle Aufbau des Channel TEDS Datenblocks mit den sechs Subblöcken Datenstrukturbezogener Subblock, Transduktorbezogener Subblock, Datenkodierungsbezogener Subblock, Zeitverhaltenbezogener Subblock, Ereignisbezogener Subblock und Datenintegritätsbezogener Subblock wird unter Bezugnahme auf die jeweils relevanten Datenfelder in Tabelle 9 zusammenfassend dargestellt.

Tabelle 9 Struktur des Channel TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 7	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet u. a. Angaben zur Länge des Channel TEDS sowie zu gegebenenfalls vorhandenen industriespezifischen TEDS-Erweiterungen für den jeweiligen Channel.
8 bis 13	Transduktor-bezogener Subblock	In diesem Subblock werden der Channel-Typ (z. B. Sensor, Aktor, Ereignissensor) und die für die Datenauswertung benötigten Grundinformationen wie oberer und unterer Grenzwert sowie die physikalische Einheit angegeben.
14 bis 20	Datenkodierungs-bezogener Subblock	Dieser Subbock wird für die Beschreibung des Datenmodells, das ausschließlich Ganz- und Gleitkommazahlen umfasst, verwendet. Die Bereitstellung mehrerer Datenpunkte pro Anfrage wird unterstützt. Hierzu kann das Inkrement zweier Datenpunkte (z. B. der zeitliche Abstand) näher spezifiziert werden.
21 bis 28	Zeitverhalten-bezogener Subblock	Die in diesem Subblock beinhalteten Datenfelder beschreiben das Zeitverhalten des Channels. Diese Angaben sind für die Kommunikation zwischen NCAP und STIM von Relevanz. Beispielsweise kann die Erkennung eines Time-Out durch ein NCAP mittels Abgleich von Wartezeit und erwarteter Reaktionszeit („Channel Update Time“) erfolgen.
29	Ereignisbezogener Subblock	Das „Event sequence options field“ ist nur dann von Bedeutung, wenn der betreffende Channel einen Ereignissensor repräsentiert. In diesem Fall kann über das Datenfeld spezifiziert werden, ob das STIM mögliche Inkonsistenzen hinsichtlich der Parametrisierung detektieren kann. Eine Inkonsistenz liegt beispielsweise vor, wenn bei einem analogen Ereignissensor die obere und untere Schranke so gesetzt werden, dass mindestens eine außerhalb des im Channel TEDS spezifizierten Wertebereichs (siehe Transduktorbezogener Subblock) liegt. Obere und untere Schranke des Ereignissensors dient hierbei der Erwirkung eines hysteresebehafteten Ereignisauslöseverhaltens.
30	Datenintegritäts-bezogener Subblock	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Im Calibration TEDS Datenblock sind Angaben mit Kalibrierungsrelevanz beinhaltet. Kalibrierung im Sinne des Standards IEEE 1451.2 ist die Anwendung einer mathematischen Funktion auf Daten, die zwischen NCAP und STIM ausgetauscht werden. Diese mathematische Funktion setzt multivariable Eingangsdaten mit einem Zielwert in Beziehung. Die Quelle der Eingangsdaten und das Ziel der Ergebnisdaten variieren in Abhängigkeit des Transduktortyps. Es sind zwei Fälle möglich:

1. Bei einem Sensor wird das Eingangsdatum des zugehörigen Channels STIM-seitig (z. B. ein Spannungswert) bezogen und das Ergebnis (z. B. die dem Spannungswert entsprechende Temperatur) dem NCAP bereitgestellt.
2. Bei einem Aktor wird die Kalibrierungsfunktion hingegen auf das NCAP-seitige Eingangsdatum des zugehörigen Channels (z. B. gewünschte Leuchtstärke einer Lampe) angewandt und das Ergebnis (z. B. die der Leuchtstärke entsprechende Spannung) vom STIM weiterverarbeitet.

Die Eingangsdaten können in beiden Fällen durch weitere Channels (Aktoren oder Sensoren) ergänzt werden. Abbildung 37 zeigt die beschriebenen Funktionsweisen der Sensor- und Aktorkalibrierung.

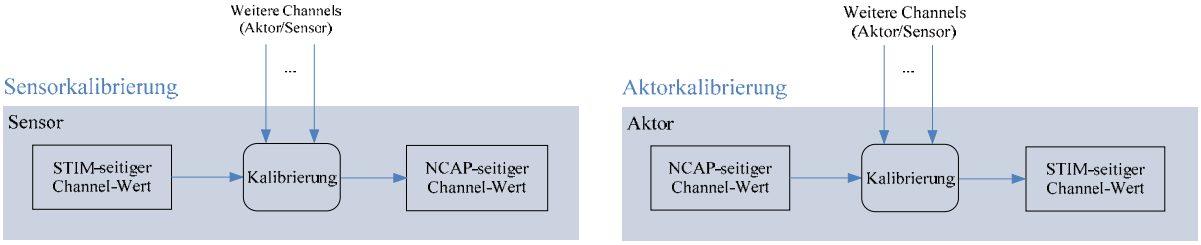


Abbildung 37 Funktionsweise der Sensor- und Aktorkalibrierung. Die Sensorkalibrierung zeichnet sich dadurch aus, dass der vom Smart Transducer Interface Module (STIM) bereitgestellte Wert verarbeitet und dann an den Network Capable Application Processor (NCAP) weitergeleitet wird. Bei der Aktorkalibrierung stellt hingegen der NCAP die Quelle der Eingangsdaten und der STIM das Ziel der Ergebnisdaten dar.

Die multinomielle Kalibrierungsfunktion f_{Kal} lautet:

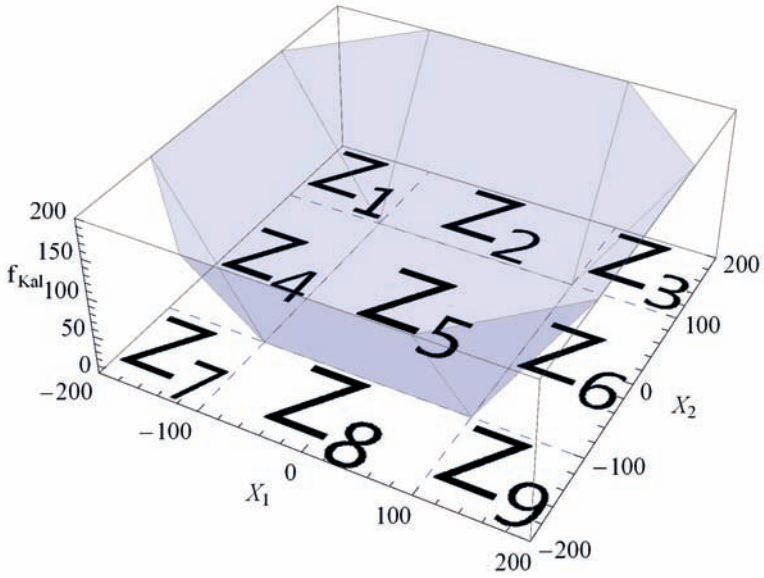
$$f_{Kal} = \sum_{i=0}^{D(1)} \sum_{j=0}^{D(2)} \dots \sum_{p=0}^{D(n)} C_{i,j,\dots,p} (X_1 - H_1)^i (X_2 - H_2)^j \dots (X_n - H_n)^p \text{ mit} \tag{2.1}$$

X_K: K-te Eingangsvariable
D(K): Grad der Eingangsvariable X_K
H_K: Offset der Variable X_K
$C_{i,j,\dots,p}$: Koeffizient des Terms $(X_1\text{-}H_1)^i(X_2\text{-}H_2)^j\dots(X_N\text{-}H_N)^p$

Für zwei Eingangsvariablen X_1 und X_2 jeweils vom Grad 1 ergibt sich der Ausdruck f_{Kal} zu $f_{Kal,1}$:

$$\begin{aligned} f_{Kal,1} &= \sum_{i=0}^{1} \sum_{j=0}^{1} C_{i,j} (X_1 - H_1)^i (X_2 - H_2)^j \\ &= C_{00} + C_{01}(X_2 - H_2) + C_{10}(X_1 - H_1) + C_{11}(X_1 - H_1)(X_2 - H_2) \end{aligned} \tag{2.2}$$

Der Grad der Eingangsvariablen kann durch Segmentierung der Wertbereiche minimiert werden. Für jede der entstehenden Zellen gilt f_{Kal} mit jeweils frei wählbaren $C_{i,j,\dots,p}$. Zudem kann für jedes Segment von X_K der Offset H_K unterschiedliche Werte annehmen. Eine gültige Ausprägung von $f_{kal,1}$ mit 3x3-Partitionierung unter Angabe der für jede Zelle Z_1 bis Z_9 gültigen Gleichung zeigt Abbildung 38.



Z_1: $f_{Kal,1} = -400+2(X_2-0)-2(X_1-0)+0(X_1-0)(X_2-0) = -2X_1+2X_2-400$
Z_2: $f_{Kal,1} = -200+2(X_2-0)-0(X_1-0)+0(X_1-0)(X_2-0) = 2X_2-200$
Z_3: $f_{Kal,1} = -400+2(X_2-0)+2(X_1-0)+0(X_1-0)(X_2-0) = 2X_1+2X_2-400$
Z_4: $f_{Kal,1} = -200+0(X_2-0)-2(X_1-0)+0(X_1-0)(X_2-0) = -2X_1-200$
Z_5: $f_{Kal,1} = 0+0(X_2-0)+0(X_1-0)+0(X_1-0)(X_2-0) = 0$
Z_6: $f_{Kal,1} = -200+0(X_2-0)+2(X_1-0)+0(X_1-0)(X_2-0) = 2X_1-200$
Z_7: $f_{Kal,1} = -400-2(X_2-0)-2(X_1-0)+0(X_1-0)(X_2-0) = -2X_1-2X_2-400$
Z_8: $f_{Kal,1} = -200-2(X_2-0)+0(X_1-0)+0(X_1-0)(X_2-0) = -2X_2-200$
Z_9: $f_{Kal,1} = -400-2(X_2-0)+2(X_1-0)+0(X_1-0)(X_2-0) = 2X_1-2X_2-400$

Abbildung 38 Kalibrierungsfunktion mit 3x3-Partitionierung. Der Partition Z_i in der x_1-x_2-Ebene wird $f_{Kal,i}$ zugeordnet.

Der Aufbau des Calibration TEDS Datenblocks mit den 3 Subblöcken Datenstrukturbezogener Subblock, Kalibrierungsbezogener Subblock und Datenintegritätsbezogener Subblock ist Tabelle 10 zu entnehmen.

Tabelle 10 Struktur des Calibration TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet die Länge des Calibration TEDS Datenblocks.
2 bis 11	Kalibrierungs-bezogener Subblock	Dieser Subblock beinhaltet die für die Kalibrierung notwendigen Informationen wie Segmenteinteilungen, Eingangsvariable X_K, deren Grade D(K), Koeffizienten $C_{i,j,\ldots,p}$ und Offsetwerte H_K (siehe f_{kal}).
12	Datenintegritäts-bezogener Subblock	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Der Identifizierung des STIM dienende textuelle Beschreibungen können über den Meta-Identification TEDS Datenblock bereitgestellt werden. Hierbei werden mehrere Sprachen unterstützt. Für jede der unterstützten Sprachen sind die Felder 4 bis 22, die identifizierungsbezogene und gruppierungsspezifische Angaben umfassen, zu wiederholen. Den Aufbau des Meta-Identification TEDS mit den 5 Subblöcken Datenstrukturbezogener Subblock, Identifikationsbezogener Subblock, Gruppierungsbezogener Subblock, Datenintegritätsbezogener Subblock I und Datenintegritätsbezogener Subblock II zeigt Tabelle 17.

Tabelle 11 Struktur des Meta-Identification TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet die Länge des Meta-Identification TEDS Datenblocks, die Anzahl der unterstützten Sprachen sowie eine Liste mit den jeweiligen Sprachcodes.
4 bis 17	Identifikations-bezogener Subblock	Zu dem Identifikationsbezogenen Subblock zählen Herstelleridentifizierung, Modellnummer, Versionsnummer, Seriennummer und Produktbeschreibung.
18 bis 21	Gruppierungs-bezogener Subblock	Den im Meta-TEDS Datenblock definierten Transduktorgruppen können in diesem Subblock Gruppenbezeichnungen zugeordnet werden.
22	Datenintegritäts-bezogener Subblock I	Die Checksumme über die Datenfelder eines Sprachblocks (Felder 4 bis 21) wird in dem datenintegritätsbezogenen Subblock I ausgewiesen.
23	Datenintegritäts-bezogener Subblock II	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Der Channel Identification TEDS Datenblock schafft die Grundlage für eine mögliche Channelidentifizierung. Die Felder des identifikationsbezogenen Datenblockes inklusive Checksumme wiederholen sich analog zum gleichnamigen Subblock des Meta-Identification TEDS Datenblocks für jede unterstützte Sprache. Zahlreiche Datenfelder wie Herstelleridentifizierung, Modellnummer etc. sind sowohl im Meta-Identification TEDS Datenblock als auch im Channel Identification Datenblock enthalten. Der Unterschied in der Bedeutung liegt in dem abweichenden Bezug. Wahlweise wird ein einzelner Channel oder ein STIM qualifiziert. Der Aufbau des Channel Identification TEDS Datenblocks mit seinen 4 Subblöcken Datenstrukturbezogener Subblock, Identifikationsbezogener Subblock, Datenintegritätsbezogener Subblock I und Datenintegritätsbezogener Subblock II ist nachfolgender Tabelle zu entnehmen.

Tabelle 12 Struktur des Channel Identification TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet die Länge des Channel Identification TEDS Datenblocks, die Anzahl der unterstützten Sprachen sowie eine Liste mit den jeweiligen Sprachcodes.
4 bis 15	Identifikations-bezogener Subblock	Identifikationsmerkmale eines Channels wie Herstelleridentifizierung, Modellnummer, Versionsnummer, Seriennummer und Channelbeschreibung werden in diesem Subblock gespeichert.
16	Datenintegritäts-bezogener Subblock I	Die Checksumme über die Datenfelder eines Sprachblocks (Felder 4 bis 15) wird in dem datenintegritätsbezogenen Subblock I ausgewiesen.
17	Datenintegritäts-bezogener Subblock II	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Die Kalibrierungsfunktion eines Channels kann textuell mit dem Calibration Identification TEDS Datenblock näher beschrieben werden. Die Speicherung identifizierungsrelevanter Angaben in mehreren Sprachen erfolgt auf dieselbe Weise wie in den bereits vorgestellten Datenblöcken Meta-Identification TEDS und Channel Identification TEDS. Im Calibration Identification TEDS Datenblock sind die Felder 4 bis 8 für jede unterstützte Sprache zu wiederholen. Der Aufbau mit den Subblöcken Datenstrukturbezogener Subblock, Identifikationsbezogener Subblock, Datenintegritätsbezogener Subblock I und Datenintegritätsbezogener Subblock II ist in Tabelle 13 dargestellt.

Tabelle 13 Struktur des Calibration Identification TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet die Länge des Calibration Identification TEDS Datenblocks, die Anzahl der unterstützten Sprachen sowie eine Liste mit den jeweiligen Sprachcodes.
4 bis 7	Identifikations-bezogener Subblock	Der mit dem Channel verbundenen Kalibrierungsfunktion kann in diesem Subblock eine Bezeichnung zugeordnet werden.
8	Datenintegritäts-bezogener Subblock I	Die Checksumme über die Datenfelder eines Sprachblocks (Felder 4 bis 7) wird in dem datenintegritätsbezogenen Subblock I ausgewiesen.
9	Datenintegritäts-bezogener Subblock II	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Applikationsspezifische Daten mit Bezug auf das STIM oder alternativ mit Bezug auf einzelne Channels, wozu z. B. Wartungsintervalle zählen, können mittels End-Users' Application Specific TEDS Datenblock in mehreren Sprachen verwaltet werden. Die Felder 4 bis 8 sind für jede unterstützte Sprache zu wiederholen. Tabelle 17 zeigt die Datenstruktur des End-Users' Application-Specific TEDS Datenblocks mit den Subblöcken Datenstrukturbezogener Subblock, Applikationsbezogener Subblock, Datenintegritätsbezogener Subblock I und Datenintegritätsbezogener Subblock II.

Tabelle 14 Struktur des End-Users' Application-Specific TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Datenstruktur-bezogener Subblock	Dieser Subblock beinhaltet die Länge des End-Users' Application-Specific TEDS Datenblocks, die Anzahl der unterstützten Sprachen sowie eine Liste mit den jeweiligen Sprachcodes.
4 bis 7	Applikations-bezogener Subblock	In diesem Subblock können Endnutzerdaten bestehend aus bis zu 65.528 Zeichen applikationsspezifisch verwaltet werden.
8	Datenintegritäts-bezogener Subblock I	Die Checksumme über die Datenfelder eines Sprachblocks (Felder 4 bis 7) wird in dem datenintegritätsbezogenen Subblock I ausgewiesen.
9	Datenintegritäts-bezogener Subblock II	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Mit dem Ziel der Erweiterbarkeit sieht der Standard IEEE 1451.2 den Einsatz von Generic Extension TEDS vor. Hiermit können Strukturen abgebildet werden, die über die oben beschriebenen Standard TEDS hinausgehen. Dies ermöglicht die Implementierung industriespezifischer Erweiterungen wie beispielsweise die Schaffung einer Struktur mit typischen Merkmalen von Beschleunigungssensoren.

Unter Verwendung sogenannter Extension Keys werden die zu einem STIM gehörigen Generic Extension TEDS im Meta TEDS Datenblock und die zu einem Channel gehörigen Generic Extension TEDS im Channel TEDS Datenblock referenziert. Mit einer Extension Key ist gleichzeitig die für den Zugriff notwendige Funktionsnummer festgelegt. Die konkrete Struktur einer TEDS Erweiterung ist eindeutig über die Extension TEDS ID, die im Rahmen eines Erweiterungsbegehrens durch den Hersteller zu beantragen ist, bestimmt. Der geschilderte Zusammenhang ist in Abbildung 39 dargestellt.

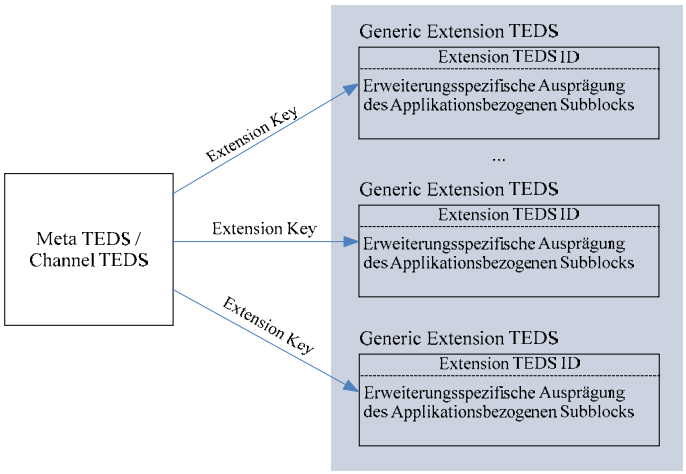


Abbildung 39 Referenzierung von TEDS Erweiterungen. Meta TEDS und Channel TEDS können mittels Extension Key auf Generic Extension TEDS verweisen. Die Zuordnung ist durch Hersteller frei wählbar. Die konkrete Struktur eines Generic Extension TEDS geht eindeutig aus dessen Extension TEDS ID hervor.

Die allgemeine Struktur eines Generic Extension TEDS Datenblocks mit den Subblöcken Datenstrukturbezogener Subblock, Applikationsbezogener Subblock und Datenintegritätsbezogener Subblock ist Tabelle 15 zu entnehmen.

Tabelle 15 Struktur des Generic Extension TEDS Datenblocks.

Datenfeld-nummer	Subblock	Beschreibung
1 bis 3	Datenstruktur-bezogener Subblock	Mit den Datenfeldern dieses Subblocks werden die Länge des Generic Extension TEDS Datenblocks, die Extension TEDS ID und die Extension TEDS Versionsnummer vorgegeben.
4 bis (N-1)	Applikations-bezogener Subblock	Dieser Subblock beinhaltet die TEDS Erweiterungsdaten in einer über die Extension TEDS ID definierten Struktur.
N	Datenintegritäts-bezogener Subblock	Zur Gewährleistung der Datenintegrität beinhaltet der letzte Subblock die Checksumme über alle vorhergehenden Subblöcke.

Konkrete Ausprägungen von Meta-TEDS, Channel TEDS sowie Calibration TEDS sind im Anhang A0 aufgeführt. Die unterschiedlichen TEDS nehmen Bezug zu einem Drucksensor der Firma Texas Instruments. Das Beispiel ist dem Paper „IEEE-P1451.2 Smart Transducer Interface Module“ entnommen[90].

2.2.5 Weitere Ansätze

Es existieren weitere Basisprotokolle, mit denen die Steuerung und Verwaltung von netzwerkfähigen Geräten möglich sind. Hierzu zählt u. a. das auf UDP basierende Simple Network Management Protocol[91] (SNMP). Das zentrale Netzwerkmanagement beruht hierbei auf der Bereitstellung einer netzwerkkomponentenspezifischen Management Information Base (MIB) durch Agenten über den UDP-Port 161. Die MIB beinhaltet in tabellarischer Form charakteristische Daten, auf die Managementstationen lesend und schreibend zugreifen können. Der direkte Aufruf von Gerätefunktionen ist hingegen nicht möglich[92].

MIB-II[93] bildet die generische Management Information Base, die von jedem zu managendem Internetknoten bereitzustellen ist. Die Datenstruktur umfasst z. B. den Systemnamen, die Systembeschreibung und TCP/IP-relevante Knoteninformationen. MIB-II umfasst alle Objekte der ersten Version MIB-I[94]. Die ergänzende Definition optionaler privater MIBs erlaubt die Berücksichtigung geräte- und herstellerspezifischer Merkmale. SNMP unterstützt die Datenübermittlung mittels Push-Mechanismus. Durch Auslösen sogenannter Traps können Agenten zuvor registrierte Managementstationen über die Änderung von MIB-Objekten informieren. Für den Empfang der hiermit assoziierten Pakete ist der UDP-Port 162 zu verwenden.

Autokonfiguration wird von SNMP nicht geboten. Ein aktuelles Netzabbild mit den jeweils angebundenen Knoten wird nicht nachgehalten. Für die Administration eines Knotens mit SNMP ist dessen momentan gültige IP-Adresse daher anderweitig zu ermitteln. Ohne den Einsatz von Discovery-Protokollen wie weiter oben beschrieben sind die Einstellungen manuell vorzunehmen. Die ausschließliche Verwendung von SNMP bei dynamisch konstituierenden Netzen ist aus diesem Grunde nicht geeignet. Für den Teilaspekt der Dienstbekanntgabe kann z. B. das Service Location Protocol (SLP) eingesetzt werden.

---

[90] Woods et al.: IEEE-P1451.2 Smart Transducer Interface Module, S. 34 ff.
[91] Case et al.: Introduction and Applicability Statements for Internet Standard Management Framework, RFC 3410.
[92] Stallings: SNMP, SNMPv2, SNMPv3, and RMON1 and 2, S. 163.
[93] McCloghrie/Rose: Management Information Base for Network Management of TCP/IP-based internets: MIB-II, RFC 1213.
[94] McCloghrie/Rose: Management Information Base for Network Management of TCP/IP-based internets, RFC 1156.

In dem Kommunikationsprozess wird zwischen Verzeichnis-, Dienst- und Benutzeragenten unterschieden. Verzeichnisagenten nehmen die Registrierung von Dienstagenten entgegen und stellen die Informationen Benutzeragenten zur Verfügung. Eine Service URL besitzt folgende Form[95]:

```
service:<srvtype>://<addrspec>
```

Mit *<srvtype>* wird der Dienstanbieter – gegebenenfalls ergänzt um das zur Verwendung kommende Protokoll – beschrieben. *<addrspec>* umfasst die Hostbezeichnung und optional die durch „:“ abzutrennende Portnummer. Diese Informationen sind Grundlage von Suchanfragen. Die *<srvtype>*-basierte Suche eines registrierten Dienstes mit der Service URL

```
service:device-description:snmp://192.168.2.112:4050
```

kann beispielsweise unter Angabe von „*service:device-description*“ oder „*service:device-description:snmp*“ erfolgen. SLP ist für das Auffinden von Diensten und nicht für deren differenzierte Beschreibung bestimmt. Damit ist es zu SNMP komplementär. Es wird deutlich, dass über Jini, Zeroconf, UPnP und IEEE 1451 hinaus zwar weitere Technologien zur Realisierung verteilter Ad-Hoc-Systeme existieren. Zumeist decken die etablierten Protokolle jedoch nur Teilaspekte ab. Im Rahmen der vorliegenden Arbeit werden Ansätze mit einer auf das Mindeste beschränkten Protokollvielfalt favorisiert – funktionale Gleichwertigkeit vorausgesetzt. Aus diesem Grund sind Protokolle wie SNMP und SLP von den weiteren Überlegungen ausgenommen. Nichtsdestotrotz wird auf eine offene Gestaltung des Lösungsansatzes hingewirkt, sodass bei geänderten Rahmenbedingungen eine Protokollsubstitution mit geringem Aufwand möglich ist.

2.3 Geräteidentifikation

Die Steuerung eines Gerätes bedingt zwangsläufig dessen vorherige Auswahl durch den Nutzer. Diesbezüglich bieten sich vielfältige Gestaltungsmöglichkeiten. Vor dem Hintergrund der Konzeptionierung und Entwicklung einer generischen Lösung, die den Anforderungen an eine intuitiv nutzbare Gesamtinfrastruktur gerecht wird, werden im Rahmen der vorliegenden Arbeit zwei Optionen weiterverfolgt. Die erste Option besteht darin, dass der Nutzer eine Selektion aus einer über ein User-Interface dargebotenen Liste verfügbarer Geräte anhand deskriptiver Merkmale vornimmt. Bei einer großen Anzahl von im Verbund vorhandenen Geräten bedingt dies eine hinreichend genaue Merkmalsbeschreibung, um eindeutige Zuordnungen treffen zu können. Darüber hinaus wächst mit steigender Anzahl von Auswahlmöglichkeiten der Aufwand für den Selektionsvorgang. Im Hinblick auf diesen Aspekt kann durch die zweite Option der automatisierten Identifikation des physischen Objektes vor Ort eine Optimierung herbeigeführt werden.

Zu den wichtigsten Auto-ID-Verfahren zählen Optical Character Recognition (OCR), Radio Frequency Identification (RFID) und Barcode. Als Voraussetzung dieses Lösungsansatzes muss die unmittelbare Nähe von Nutzer und zu steuerndem Gerät gegeben sein. Tabelle 16 gibt einen Überblick über wesentliche Parameter der vorgenannten Verfahren[96]. Eine Technologiebeschreibung unter Beachtung des avisierten Einsatzes im mobilen Umfeld schließt sich an.

---

[95] Guttmann et al.: Service Location Protocol, Version 2, RFC 2608.

[96] Finkenzeller: RFID Handbuch, S. 8.

Tabelle 16 Vergleich verschiedener Auto-ID-Systeme.

Parameter	OCR	RFID	Barcode
Typische Datenmenge/Byte	1...100	16...64 k	1...100
Datendichte	gering	sehr hoch	gering
Maschinenlesbarkeit	gut	gut	gut
Lesbarkeit durch Personen	einfach	unmöglich	bedingt
Einfluss von Schmutz/Nässe	sehr stark	kein Einfluss	sehr stark
Einfluss von (opt.) Abdeckung	totaler Ausfall	kein Einfluss	totaler Ausfall
Einfluss von Richtung und Lage	gering	kein Einfluss	gering
Abnutzung, Verschleiß	bedingt	kein Einfluss	bedingt
Anschaffungskosten Elektronik	mittel	mittel	sehr gering
Betriebskosten (z. B. Drucker)	gering	keine	gering
unbefugtes Kopieren/Ändern	leicht	unmöglich	leicht
Lesegeschwindigkeit (incl. Handhabung des Datenträgers)	gering ~ 3 s	sehr schnell ~ 0,5 s	gering ~ 4 s
Maximale Entfernung zwischen Datenträger und Lesegerät	< 1 cm (Scanner)	0...5 m, Mikrowelle	0...50 cm

2.3.1 OCR

Die Erkennung von Klarschrifttexten durch elektronische Datenverarbeitungssysteme wird als Optical Character Recognition bezeichnet. Die ersten OCR-Systeme, bestehend aus Schreib-Lese-Gerät sowie dazugehöriger Software, wurden bereits in den 60er Jahren eingeführt. Hiermit einher ging die Entwicklung maschinenlesbarer Schriften mit dem Ziel einer schnellen und rechenarmen Unterscheidbarkeit der jeweiligen Zeichen. Hierzu zählen zum Beispiel OCR-A (DIN 66008[97], ISO 1073-1[98]), OCR-B (DIN 66009[99], ISO 1073-2[100]) und OCR-H (DIN 66225[101]).

Weit fortgeschrittene Algorithmen sowie die gestiegene Leistungsfähigkeit moderner Computersysteme ermöglichen die Abkehr von OCR-spezifischen hin zu Standard-Schriftsätzen. Selbst bei handgeschriebenen Texten werden mittlerweile hohe Erkennungsraten erzielt, so beispielweise bei der automatisierten Auswertung von Adressangaben im Logistikbereich sowie beim automatisierten Einlesen von handschriftlich ausgefüllten Überweisungsträgern im Bankenbereich.

Ein großer Vorteil gegenüber den Technologien Barcode und RFID besteht in der Möglichkeit der visuellen Informationserfassung direkt durch den Menschen. Diese Technologie ist für den Einsatz im mobilen Umfeld potentiell geeignet, da mit einer handelsüblichen – in einem Mobilfunkgerät bereits integrierten – Kamera die für die Informationserfassung notwendige Scaneinheit bereitgestellt wird.

---

[97] Din: Schrift A für die maschinelle optische Zeichenerkennung, DIN 66008.
[98] Iso: Alphanumeric character sets for optical recognition – Part 1: Character set OCR-A, ISO 1073-1.
[99] Din: Schrift B für die maschinelle optische Zeichenerkennung, DIN 66009.
[100] Iso: Alphanumeric character set for optical recognition – Part 2: Character set OCR-B, ISO 1073-2.
[101] Din: Schrift H für die maschinelle optische Zeichenerkennung, DIN 66225.

Für die Einbindung der notwendigen Erkennungslogik stellt zum Beispiel die Firma ABBYY ein funktionsreiches Entwicklerkit zur Verfügung. Mit der Mobile OCR Engine 3.0 können mobile Anwendungen unter anderem für die Zielsysteme Windows Mobile, Symbian OS und Linux entwickelt werden. Hierbei werden gemäß Herstellerangabe 58 Sprachen unterstützt[102]. Für die Erstellung von OCR-Labels können Standard-Textverarbeitungsprogramme wie MS Word genutzt werden.

2.3.2 RFID

Die Bezeichnung RFID (Radio Frequency Identification) ist auf den Mechanismus der Objektidentifizierung mittels elektromagnetischer Wellen zurückzuführen. Die kontaktlose Datenübertragung zwischen den Systemkomponenten Transponder sowie Schreib-Lese-Gerät beschränkt sich nicht wie der Begriff RFID suggeriert auf den Radiofrequenzbereich, sondern erfolgt in einem weitaus breiterem Frequenzspektrum, beinhaltend:

- Low Frequency (LF): 30 kHz – 300 kHz
- Radio Frequency (RF)/High Frequency: 3 MHz – 30 MHz
- Ultra High Frequency (UHF): 300 MHz – 3 GHz
- Mikrowelle: > 3 GHz

Typische Transponder-integrierende Lösungen, die Informationen über das zu identifizierende Trägerobjekt bereitstellen, sind zum Beispiel Klebeetiketten, Plastikkarten, Glasröhrchen und Token. In Bezug auf die Stromversorgung wird differenziert in aktive und passive Transponder. Aktive Transponder sind dadurch gekennzeichnet, dass sie entweder eine Batterie für die Vollstromversorgung oder eine Stützbatterie für die Teilstromversorgung enthalten. Tendenziell können mit aktiven Transpondern höhere Reichweiten als mit passiven erzielt werden. Diese beziehen die benötigte Energie bei kapazitiver Kopplung aus elektrischen, bei induktiver Kopplung aus magnetischen Wechselfeldern, die von dem Schreib-Lese-Gerät generiert werden. Für den mobilen Bereich sind sowohl separate Schreib-Lese-Geräte als auch integrierte Lösungen, die mobile Endgeräte wie Mobiltelefone oder PDAs um RFID-Funktionalität erweitern, erhältlich. Abbildung 40 zeigt den als iCarte bezeichneten RFID-Aufsatz der Firma Wireless Dynamics für das iPhone[103].

Abbildung 40 iCarte der Firma Wireless Dynamics. Mit dem Aufsatz wird ein iPhone zu einem vollwertigen RFID-Schreib-Lese-Gerät.

In der Regel umfassen die Produktpakete Programmbibliotheken, die die Einbettung der RFID-Technologie in eigene Anwendungen ermöglichen.

[102] http://www.abbyy.de/mobileocr/benefits, zugegriffen am 16.11.2009
[103] Abbildung von http://www.icarte.ca, zugegriffen am 09.04.2010

2.3.3 Barcode

Die der Informationskodierung dienenden Strichcodes werden als Barcodes bezeichnet. Zahlreiche genormte und herstellerspezifische Varianten sind im Einsatz. Ein wesentliches Klassifizierungsmerkmal ist die für die Kodierung genutzte Anzahl der Dimensionen. Bei der parallelen Anordnung von Strichen und Lücken beinhaltet nur die hierzu orthogonal stehende Dimension, die Leserichtung, codierte Informationen. Der informationelle Gehalt ist ausschließlich durch die Breite der Striche und Lücken gegeben, nicht durch ihre Höhe. Sehr verbreitet in dieser Klasse ist der European Article Number (EAN) Code, bei dem eine 13-oder 8-stellige Artikelnummer kodiert wird.

Abbildung 41 EAN13-Code[104]. Durch Variation der Breite von Strichen und Lücken werden Informationen kodiert. Somit liegt ein 1-dimensionaler Code vor.

Der EAN-Code[105] stellt eine Weiterentwicklung des US-amerikanischen Universal Product Code (UPC) dar. UPC ist eine Untermenge des EAN-Codes und somit kompatibel zu diesem[106]. Eine höhere Informationsdichte kann mit zweidimensionalen Strichcodes erzielt werden. Hierbei werden Informationen in horizontaler und vertikaler Richtung kodiert. Eine Variante dieser Klasse ist der als PDF417 bezeichnete Code, der in der Norm ISO 15438[107] spezifiziert ist. PDF steht in diesem Zusammenhang für Portable Data File, die Verwendung der Zahl 417 ist darauf zurückzuführen, dass die im Symbol (siehe Abbildung 42) enthaltenen Datenzeichen, jeweils bestehend aus 4 Balken und Lücken, in 17 Module untergliedert sind.

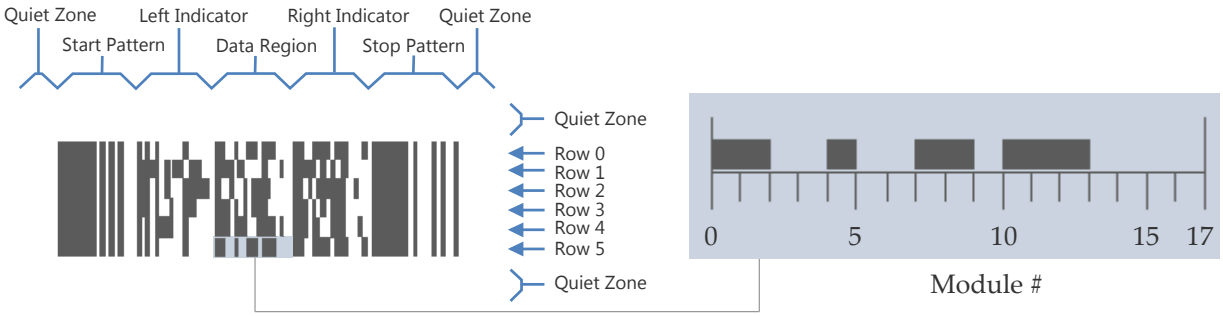


Abbildung 42 PDF417-Symbol[108]. Es umfasst mehrere horizontal und vertikal angeordnete Datenzeichen. Jedes Datenzeichen besteht aus 4 Balken sowie Lücken und ist in 17 Module untergliedert. Im Gegensatz zu EAN13 liegt ein 2-dimensionaler Code mit höherer Informationsdichte vor.

---

[104] Barcode des Buches „Multimedia Kommunikation“ (Froitzheim)
[105] Iso/Iec: Information technology – Automatic identification and data capture techniques – Bar code symbology specification – EAN/UPC, ISO/IEC 15420.
[106] Virnich/Posten: Handbuch der codierten Datenträger, S. 31.
[107] Iso/Iec: Information technology – Automatic identification and data capture techniques – PDF417 bar code symbology specification, ISO/IEC 15438.
[108] linker Teil der Abbildung von http://www.easesoft.net/PDF417.html, zugegriffen am 16.11.2009.

Grundsätzlich verfügt jedes PDF417-Symbol über zwei Zeichen zur Fehlererkennung. Zur Realisierung der Fehlerkorrektur werden insgesamt neun Sicherheitsstufen unterstützt. Die Größe eines Symbols wächst hierbei mit der Höhe der Sicherheitsstufe[109]. Ebenfalls starke Verbreitung gefunden hat der zweidimensionale QR (Quick Response) Code[110].

Genau wie die OCR-Technologie ist auch die Barcode-Technologie für den Einsatz im mobilen Umfeld geeignet. Ein Barcode kann analog mit einer handelsüblichen – gegebenenfalls in einem Mobilfunkgerät bereits integrierten – Kamera optisch erfasst werden. Anwendungen für die Informationskodierung und -dekodierung unter Nutzung von Barcodes stehen für alle gängigen Betriebssysteme zur Verfügung. Alternativ können für die Kodierung Online-Dienste in Anspruch genommen werden.

2.4 Audio- und Video-Streaming

Die Möglichkeit der Zustandsüberwachung von über das Internet steuerbaren Geräte mit dem Ziel der Prüfung auf erwartungskonformes Geräteverhalten und der gegebenenfalls erforderlichen Einleitung regelungstechnischer Maßnahmen ist in zahlreichen Anwendungsdomänen unabdingbar. Für die Realisierung eines audiovisuellen Rückkanals eignet sich die Streaming-Technologie, mit der eine kostengünstige Überwachungsinfrastruktur etabliert werden kann. Zu den wesentlichen Merkmalen der Streaming-Technologie zählt, dass die Nutzung multimedialer Inhalte bereits während der Übertragung erfolgt und eine Kopie der für den Abspielvorgang benötigten Daten ausschließlich zwischengespeichert wird. Streaming wird klassifiziert in Progressive und Realtime Streaming.

Multimediale Daten werden in der Regel vor der Übertragung komprimiert, da bei heutiger Netztechnik andernfalls schnell die Grenzdurchsatzraten erreicht sind. Typischerweise kommen asymmetrische Verfahren zum Einsatz. Kennzeichen der zugrunde liegenden Codecs ist die Verschiebung des Rechenaufwandes in Richtung Kodierer zugunsten des Dekodierers[111]. Die Vielfalt der zugrunde liegenden Protokolle führt dazu, dass Inkompatibilitäten zwischen Medienserver und -client mit einer nicht vernachlässigbaren Wahrscheinlichkeit auftreten. Insbesondere beim Einsatz heutiger mobiler Endgeräte, die im Allgemeinen nur eine sehr begrenzte Anzahl von Protokollen und Formaten unterstützen, kommt es häufig zu Fehlfunktionen.

Neben theoretischen Betrachtungen der Streaming-Technologie werden im Folgenden marktgängige Streaming-Server und -Clients mit ihren wesentlichen funktionalen Parametern dargestellt. Die Ausführungen bilden die Grundlage für die bedarfsgerechte Auswahl von Produktlösungen für den Einsatz im Bereich des Mobile Computing.

2.4.1 Progressive Streaming

Erfolgt die On demand-Wiedergabe einer Mediendatei, bevor diese in ihrer Gesamtheit übertragen wurde und wird hierbei kein Abgleich zwischen Übertragungs- und Bitrate des Streams vorgenommen, so wird dies als Progressive Streaming bezeichnet. Gängige zum Einsatz kommende Protokolle sind FTP und HTTP. Beide Transportprotokolle basieren auf TCP und eignen sich infolgedessen ausschließlich für Unicast-Kommunikationsbeziehungen. Eine unterbrechungsfreie Wiedergabe kann erreicht werden, indem auf dem Client ein hinreichend großer Puffer bereitgestellt wird.

---

[109] Jesse/Rosenbaum: Barcode, S. 133.
[110] Jiang/Ma/Chen: Mobile Multimedia Processing, S. 40.
[111] Austerberry: The Technology of Video and Audio Streaming, S. 153 f.

Die Verzögerung zwischen Übertragungs- und Wiedergabestart wächst mit der Diskrepanz zwischen Übertragungs- und Bitrate. Progressive Streaming ist nicht für die Übertragung von Echtzeitdaten geeignet. Zu den weiteren Charakteristika des Progressive Streaming zählt, dass die Mediendatei beginnend mit dem Offset 0 des Datenbereichs fortlaufend übertragen wird. Der Zugriff ist damit sequentiell. Die erste – zum Zeitpunkt t_0 den Client erreichende – Dateneinheit ist b_1, die auch den Beginn der zu übertragenden Datei markiert (siehe Abbildung 43). Der Datentransfer ausgewählter Teilbereiche ist hingegen nicht möglich.

Vorwärts- und Rückwärtsspulen ist somit auf die im Puffer befindlichen bzw. bereits zwischengespeicherten Daten beschränkt. Sofern ein Gerät über wenig Speicherkapazität verfügt, stellt dies einen gewichtigen Nachteil dar. Bei Volumentarifen werden im Falle partieller Medienanspruchnahme in Ermangelung des selektiven Datentransfers unnötig hohe Kosten verursacht.

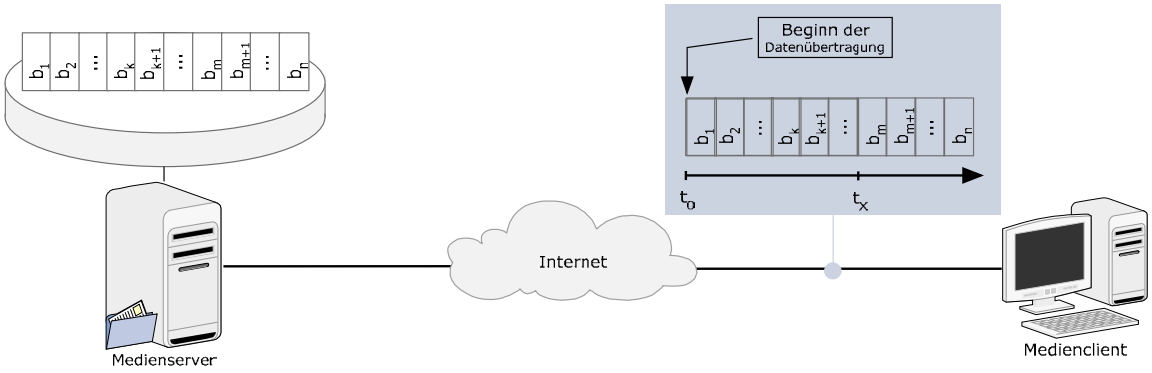


Abbildung 43 Progressive Streaming. Der Datenbereich wird fortlaufend übertragen und beginnt mit der ersten Einheit b_1. Damit ist clientseitig die gezielte Wiedergabe eines Teilabschnitts auf die im Puffer befindlichen Daten beschränkt.

2.4.2 Realtime Streaming

Erfolgt die Wiedergabe von Echtzeitinhalten wie z. B. die Live-Übertragung einer Mediendatei oder einer Webcam-Aufnahme und wird hierbei ein Abgleich zwischen Übertragungs- und Bitrate vorgenommen, so wird dies als Realtime Streaming bezeichnet. Die adaptiven Client-Server-Verbindungsparameter sind die Grundvoraussetzung für eine unterbrechungsfreie Wiedergabe bei gleichzeitiger Minimierung der Puffergröße sowie der Anfangsverzögerung. Werden Echtzeitdaten für eine im Vorfeld nicht definierte Dauer von einem Client empfangen, so ist nur dann eine unterbrechungsfreie Wiedergabe möglich, wenn die Übertragungsrate größer als die Bitrate ist. Im Gegensatz zum Progressive Streaming ist Realtime Streaming dadurch gekennzeichnet, dass bei der On demand-Wiedergabe der Datentransfer ausgewählter Teilbereiche möglich ist.

Im Folgenden wird konkret Bezug genommen zu dem Zusammenwirken der Standard-Protokolle Real Time Streaming Protocol (RTSP), Real Time Transport Protocol (RTP) und RTP Control Protocol (RTCP), um exemplarisch die wesentlichen Aspekte des Realtime Streaming näher zu erläutern[112]. Abbildung 44 dient der Veranschaulichung.

[112] Schulzrinne/Rao/Lanphier: Real Time Streaming Protocol, RFC 2326.

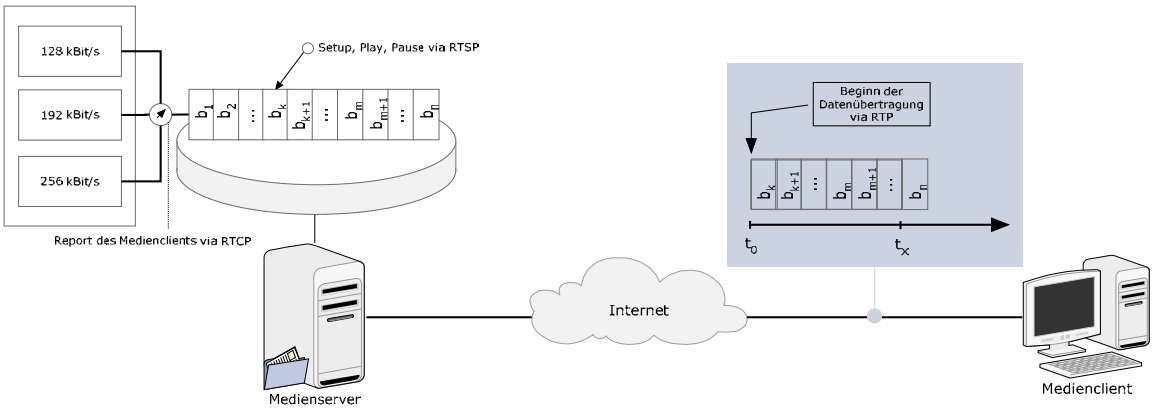


Abbildung 44 Realtime Streaming. Die Auswahl von Teilbereichen der vom Medienserver bereitgestellten Daten ist mit dem Realtime Streaming Protocol (RTSP) möglich. Vorliegend fordert der Medienclient Dateneinheiten beginnend mit b_k an. Der Datentransport erfolgt über RTP.

Mit RTSP werden die Sitzung steuernde Funktionen wie Auswählen einer Medienquelle (Describe), Festlegen des Transporteinstellungen (Setup), Initiieren (Start), Anhalten (Pause) und Beenden (Teardown) des Abspielvorgangs ausgeführt. Im Beispiel wird RTSP genutzt, um die Mediendatei beginnend mit Dateneinheit b_k anzufordern. Nutzdaten können mit RTP übertragen werden. Hierbei handelt es sich um ein Transportprotokoll, das nicht an ein bestimmtes Medienformat gebunden ist. Es eignet sich unter anderem für die Übertragung von WMA-, WMV- und 3GP-kodierten Inhalten. Typischerweise verwendet RTP in den darunterliegenden Kommunikationsschichten in Abhängigkeit der zwischen Sender und Empfänger gebotenen Möglichkeiten UDP oder TCP. Generell gilt, dass UDP-basierte gegenüber TCP-basierter Kommunikation vorzuziehen ist, da diese zwei wesentliche Vorteile bietet:

1. UDP unterstützt die Verteilung von IP-Multicast-Datagrammen an Empfängergruppen. Hierdurch kann eine erhebliche Netzwerklastreduzierung herbeigeführt werden.
2. UDP erlaubt eine Overhead-arme Datenübertragung – eine Fehlerkorrektur wird überhaupt nicht, eine Fehlererkennung nur bei Bedarf durchgeführt, das Eintreffen von UDP-Dateneinheiten beim Empfänger wird nicht bestätigt und die Kommunikation erfolgt verbindungslos. Auf diese Weise können Request-Response-Verzögerungen sowie Übertragungsverzögerungsschwankungen reduziert werden.

Unter Verwendung des RTCP werden im Rahmen der Streaming-Session zwischen Sender und Empfänger in regelmäßigen Abständen Reports übermittelt. Diese Reports beinhalten zustandsbeschreibende Größen wie erzielte Übertragungsrate und Staucharakteristika als Grundlage für die Festlegung der im weiteren Verlauf zu verwendenden Verbindungsparameter. Zumeist werden die RTSP-, RTP- und RTCP-Datenströme getrennt voneinander übertragen. Es besteht ebenfalls die Möglichkeit, RTP und RTCP in den RTSP-Datenstrom mit einzubinden (RTP over RTSP)[113]. Kommt eine restriktive Firewall zum Einsatz – zum Beispiel eine Application-Layer-Firewall, die nur FTP- und HTTP-Datenpakete passieren lässt – ist die Protokollvariante RTSP/RTP over HTTP, bei der der gesamte Datenverkehr mittels HTTP-Tunneling übertragen wird, eine aussichtsreiche Option.

[113] Schulzrinne et al.: RTP: A Transport Protocol for Real-Time Applications, RFC 1889.

Es gilt zu beachten, dass beim Streaming von RA- und RV-kodierten Inhalten RTSP in der Regel mit dem proprietären Transportprotokoll RDT der Firma RealNetworks – und nicht mit RTP – zusammenwirkt[114]. Ein weiteres sehr verbreitetes Realtime Streaming Protokoll ist Microsoft Window Media Server (MMS) Protocol[115]. Konzeptuell gibt es zahlreiche Parallelen zu dem beschriebenen Protokolltrio RTSP/RTP/RTCP. Zunächst werden mittels MMS die Sitzungsparameter festgelegt, hier kann ebenfalls zwischen UDP- und TCP-basierter Kommunikation auf Transportschichtebene gewählt werden (MMSU bzw. MMST) – mit den damit verbundenen oben bereits erläuterten Implikationen. Bei MMS werden analog zu RTCP Berichte über Quality of Service (QoS) in regelmäßigen Abständen generiert, um eine optimale Belegung von Verbindungsparametern vornehmen zu können. Da RTSP/RTP/RTCP das von Microsoft bevorzugte Streaming-Protokoll ist, wird bei aktuellen Versionen der Windows-Media-Player-Reihe der Anfrage einer mit dem Präfix „mms:“ eingeleiteten URL mit einem Protokoll-Rollover begegnet. Dabei wird versucht, für den Kommunikationsprozess RTSP/RTP/RTCP, MMS und Windows Media Streaming Protocol (WMSP)[116] zu nutzen. Die Reihenfolge der Nennung entspricht der Reihenfolge bei der Aushandlung des zu verwendenden Protokolls. Zuletzt genanntes Windows Media Streaming Protocol zeichnet sich dadurch aus, dass die Übertragung mittels HTTP-Tunneling erfolgt. Wie bei RTSP/RTP over HTTP kann bei WMSP in der Regel trotz Einsatz restriktiver Firewalls Streaming durchgeführt werden.

HTML5 erlaubt die Integration von Audio- und Videodaten in Webseiten. Dem Standard entsprechende Browser geben die Inhalte nativ wieder. Unter Nutzung des zu den Tags <*audio*> und <*video*> gehörigen API kann mittels Skriptsprachen die Steuerung vorgenommen werden. Hierzu zählt u. a. die Angabe von Wiedergabebereichen.

Wie weiter oben ausgeführt sind Server, die ausschließlich HTTP/1.0 implementieren, für den selektiven Datentransport und somit für Realtime Streaming ungeeignet. Erst mit Erweiterungen wie z. B. CGI oder sogenannten HTTP Range Requests in HTTP/1.1 ist diese Einschränkung aufhebbar. Grundlage hierbei sind ergänzende Schlüssel-Werte-Paare in URI-Queries respektive zusätzliche Header-Felder.

Bei Youtube wird der Zeitoffset auf Grundlage der querybasierten Variante gesetzt. Damit URLs wie *http://www.youtube.com/watch?v=CzTbkP63JBg#t=3m00s* tatsächlich zur partiellen – im Beispiel bei Minute drei beginnenden – Datenübertragung führen, muss das zur Anwendung kommende Format vom Medienserver interpretiert werden können.

Die W3C-Arbeitsgruppe "Media Fragments" befasst sich mit der Standardisierung der hier skizzierten Verfahrensweisen. Die diesbezügliche Ausarbeitung hat den Status Working Draft[117]. Streaming mittels HTML5 ist derzeit ausschließlich mit On demand-Inhalten möglich. Diese werden bei den weiter unten genannten Beispielen (siehe Tabelle 18) von einem Apache HTTP Server bereitgestellt. Der abrufende Chrome-Browser unterstützt hierbei den Videocodec H.264 sowie den Audiocodec AAC[118]. Da HTML5 keine Bitratenanpassung vorsieht, sind die Kriterien für Realtime Streaming nur bedingt erfüllt.

---

[114] Lee: Scalable Continuous Media Streaming Systems, S. 114.
[115] Microsoft: Microsoft Media Server (MMS) Protocol Specification.
[116] Microsoft: Windows Media HTTP Streaming Protocol Specification.
[117] Troncy et al: Media Fragments URI 1.0.
[118] Lubbers/Albers/Salim: Pro HTML5 Programming, S. 69.

Im Gegensatz zu HTML5 behandelt der von Apple eingereichte IETF-Draft „HTTP Live Streaming“[119] auch die Übertragung von Live-Inhalten auf Basis von HTTP. Das Verfahren beruht auf der Segmentierung von Medieninhalten, die anschließend über herkömmliche HTTP-Server verteilt werden. Die Verweise auf die einzelnen Segmente sind hierbei in einer Wiedergabeliste aufgeführt. Entsprechend der Linkabfolge werden clientseitig die Segmente nahtlos zusammengefügt. Die Wiedergabeliste muss im – um mehrere Tags erweiterten – M3U-Format[120] vorliegen. Sie ist vom Client in regelmäßigen Abständen auf Aktualität zu prüfen, sofern sie nicht mit dem das Ende markierenden Eintrag *#EXT-X-ENDLIST* abschließt. Auf diese Weise können auch Live-Inhalte rekonstruiert werden. Mediensegmente sind als MPEG-2-Transport Streams mit der Endung *.ts* zu speichern. HTTP Live Streaming setzt gegenwärtig den Videocodec H.264 und den Audiocodec AAC voraus. Als Segmenter eignet sich u. a. die Anwendung Helix Universal Server v14, auf die im Folgekapitel „Streaming-Server und -Clients“ eingegangen wird.

2.4.3 Streaming-Server und -Clients

Für die Bereitstellung von Medieninhalten über Progressive Streaming werden handelsübliche FTP- oder HTTP-Server eingesetzt, bei deren Einrichtung keine streamingspezifischen Besonderheiten zu beachten sind. Der FTP- bzw. HTTP-basierte Abruf wird von den meisten gängigen Medienplayern wie WMP, QuickTime und Realplayer unterstützt. Für die Bereitstellung von Medieninhalten über Realtime Streaming sind Spezialanwendungen der Kategorien Live Content Enkodierer und Streaming-Server einzusetzen. Typischerweise liefert ein Live Content Enkodierer einen kontinuierlichen Live-Datenstrom aus frei wählbaren Quellen wie z. B. Mediendateien, Kameras oder Mikrofonen an einen Streaming-Server, der wiederum die Schnittstelle zu Streaming Clients darstellt.

Auf manche Live Content Enkodierer können Clients direkt zugreifen. Dies ist z. B. bei Einsatz des Produkts Windows Media Encoder 9 Series und Nutzung von WMSP der Fall. Die Zuständigkeit für die Verwaltung und Bereitstellung von On demand-Inhalten liegt beim Streaming-Server. Die Interaktion von Live Content Enkodierer, Streaming-Server und -Client ist in Abbildung 45 visualisiert.

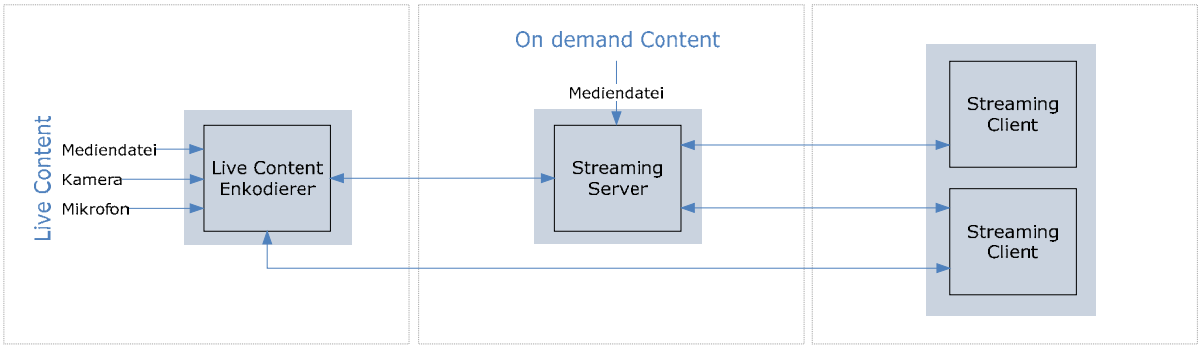


Abbildung 45 Interaktion von Live Content Enkodierer, Streaming-Server und -Client. Der vom Live Content Enkodierer generierte Datenstrom wird über einen Streaming-Server veröffentlicht und somit Streaming-Clients zur Verfügung gestellt. Manche Live Content Enkodierer unterstützen den Direktzugriff durch Clients.

[119] Pantos/May: HTTP Live Streaming, IETF Draft 2010.
[120] Vorerst für MP3-Dateien entwickelt, Kurzform für MP3 URL.

Tabelle 17 führt gängige Live Content Enkodierer mit den von ihnen unterstützten Formaten auf und stellt die mit dem Helix Universal Server v14 für die Weiterverteilung an Clients verhandelbaren Protokolle dar.

Tabelle 17 Enkodierer für die Generierung von Live Content.

Quelle	**Mediendateien**									**Kamera/Mikrofon**		
Format	RA/ RV	3GPP	MP4 (H264)	wma/ wmv	avi	mov	MP3	swf	flv	MP4 (H264)[3)]	wma/ wmv[3)]	RA/ RV[3)]
	Mit Helix Universal Server v14 für die Weiterverteilung an Clients vereinbare Protokolle											
Helix Producer Standard v13.1	RTSP/ RDT[2)]	[1)]	RTSP/ RDT; RTSP/ RTP[2)]	[1)]	[1)]	[1)]	[1)]	[1)]	[1)]	RTSP/ RDT; RTSP/ RTP; HLS[4)]	n.u.	RTSP/ RDT
Windows Media Encoder 9 Series	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	HTTP[5)] MMS	n.u.

n.u. = nicht unterstützt

1 vor der Übertragung ist die Transkodierung in RA/RV oder MP4 erforderlich
2 RA/RV und MP4 können vor der Übertragung in das jeweils andere Format transkodiert werden, es gelten dann die entsprechenden Protokolle
3 aus dem Input-Stream generiertes Output-Format
4 HTTP Live Streaming
5 der Abruf des Multimediadatenstroms durch einen Client erfolgt direkt über Windows Media Encoder 9 Series

In Tabelle 18 sind ausgewählte von Helix Universal Server v14 unterstützte Formate für On demand-Streaming sowie Live Streaming aufgeführt. Beim Live Streaming werden die RA/RV-kodierten Inhalte durch Helix Producer Standard v13.1 und die WMV/WMA-kodierten Inhalte durch Windows Media Encoder bereitgestellt. Der Tabelle ist zu entnehmen, dass es vom jeweiligen Format und vom zur Anwendung kommenden Streaming-Client abhängig ist, über welche Streaming-Protokolle die Medieninhalte übertragen werden. Bei den Angaben wird auf gängige Formate und Streaming-Clients Bezug genommen.

Einleitend wurde erwähnt, dass es beim Einsatz heutiger mobiler Endgeräte häufig zu Komplikationen kommt. Aus Tabelle 18 lässt sicht ableiten, dass eine hohe Erfolgsquote erzielt werden kann, indem Medieninhalte durch Kombination der Streaming-Produktlösungen Helix Producer Standard v13.1, Windows Media Encoder und Helix Universal Server v14 multivalent in die Formate RV/RA, MP4 und WMV/WMA kodiert sowie multivalent über die Protokolle MMS, WMSP, RTSP/RDT, RTSP/RTP und HTTP Live Streaming[121] transportiert werden. Für die Geräteüberwachung mittels Mikrofon und Kamera kann derart das Spektrum potentiell einsetzbarer Streaming-Clients verbreitert werden. Abbildung 46 zeigt die diesem Ansatz zugrunde liegende Anordnung zentraler Komponenten.

[121] Setzt als Videokodierung H.264 und als Audiokodierung AAC voraus

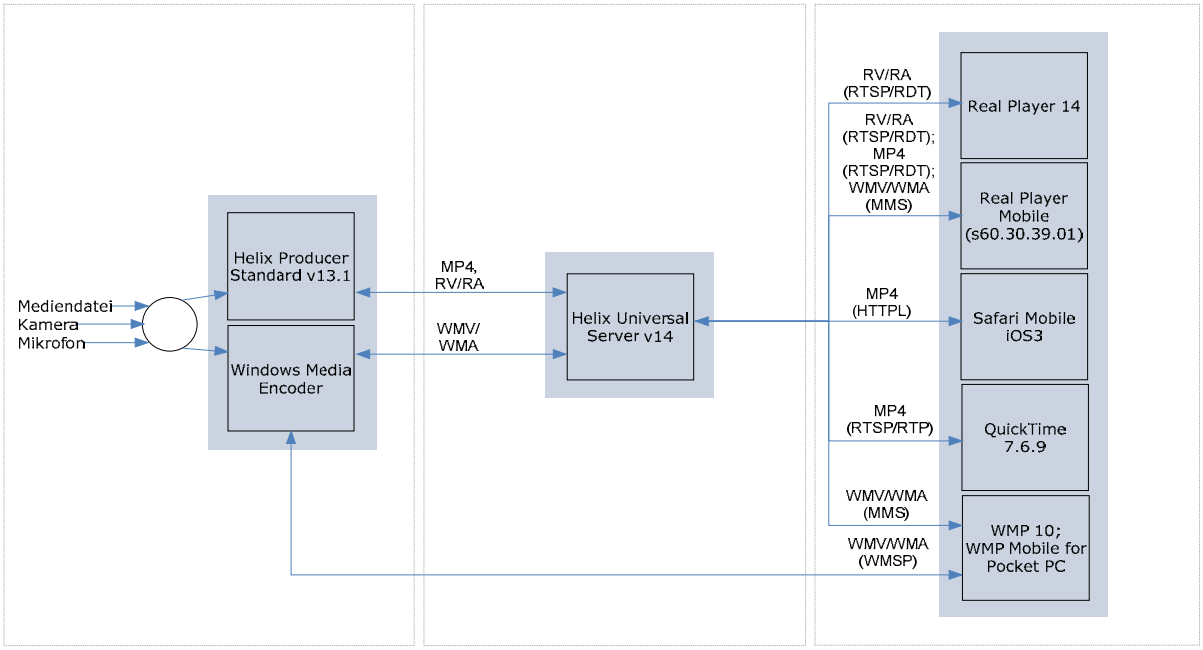


Abbildung 46 Anordnung zentraler Komponenten bei multivalentem Live Streaming. Die Bereitstellung von Medieninhalten über MMS, WMSP, RTSP/RDT, RTSP/RTP wird realisiert, indem die Live Content Enkodierer Helix Producer Standard v13.1 und Windows Media Encoder sowie der Streaming-Server Helix Universal Server v14 zum Einsatz kommen.

Tabelle 18 Realtime Streaming von On demand- und Live-Inhalten. Als Live Content Enkodierer dienen Helix Producer Standard v13.1 und Windows Media Encoder. Die Bereitstellung für gängige Streaming Clients erfolgt mittels Helix Universal Server v14.

n.u. = nicht unterstützt

Art des Streaming	**On demand Streaming von Mediendateien**							**Live Streaming (Quelle ist Live Content Enkodierer)**		
Format	RA/RV	MP4	WMV/ WMA	MOV	MP3	SWF	3GPP	RA/RV (Helix Producer Standard v13.1)	MP4 (Helix Producer Standard v13.1)	WMV/ WMA (Windows Media Encoder)
					Beispiel-Input					
Codec	RV 10, RA 10 (Stereo, 33 kHz)	H.264, AAC (Stereo, 32 kHz)	WMV 8, WMA 8 (Stereo, 32 kHz)	H.264, MP4A QDesign (Stereo, 22,05 kHz)	MPEG Audio Layer 3 (Stereo, 44,1 kHz)	Flash 4, keine Tonspur	H.264, AAC (Mono, 44,1 kHz)	RV 10, RA 10 (Mono, 22,05 kHz)	H.264, AAC (Stereo, 22,05 kHz)	WMV 9, WMA 9.1 (Mono, 32 kHz)
Max. Bitrate	150 Kbps	370 Kbps	242 Kbps	85 Kbps	192 Kbps	57 Kbps	1000 Kbps	100 Kbps	150 Kbps	273 Kbps
Auflösung	176 x 144	320 x 240	320 x 240	192 x 242	-	640 x 480	128 x 96	176 x 144	320 x 240	320 x 240
					Bereitgestellte Streaming-Protokolle des Helix Universal Server v14					
Protokoll	RTSP/RDT	RTSP/RDT; RTSP/RTP	MMS	RTSP/RTP	RTSP/RDT; RTSP/RTP	RTSP/RDT	RTSP/RDT ; RTSP/RTP	RTSP/RDT	RTSP/RDT; RTSP/RTP; HLS	MMS
					Zwischen Streaming Clients und Helix Universal Server v14 für den Beispiel-Input verwendete Streaming-Protokolle					
WMP	n.u.	n.u.	MMS	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	MMS
QuickTime	n.u.	RTSP/RTP	n.u.	RTSP/RTP	n.u.	n.u.	RTSP/RTP	n.u.	RTSP/RTP[1)]	n.u.
Real Player 11	RTSP/RDT	RTSP/RDT	MMS	RTSP/RTP[2)]	RTSP/RDT	RTSP/RDT	RTSP/RDT	RTSP/RDT	RTSP/RDT	MMS
Real Mobile	RTSP/RDT	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	RTSP/RDT	n.u.	n.u.
WMP 10 Mobile	n.u.	n.u.	MMS	n.u.	n.u.	n.u.	n.u.	n.u.	n.u.	MMS
Safari Mobile	n.u.	HLS	n.u.	HLS	n.u.	n.u.	HLS	n.u.	HLS	n.u.
Chrome (nativ)	n.u.	HTTP[3)]	n.u.	HTTP[3)]	HTTP[3)]	n.u.	HTTP[3)]	n.u.	n.u	n.u

[1] für die korrekte Wiedergabe musste in der Testumgebung innerhalb der Anwendung QuickTime der sichere GDI-Modus aktiviert sein

[2] mittels QuickTime Plugin

[3] bereitgestellt durch HTTP Server, fragmentierte Übertragung von Mediendaten mit Erweiterung möglich

Kapitel 3

Petrinetze

Dem Verhalten eines isolierten Gerätes als auch dem Verhalten eines Gerätes im Kontext eines Systemverbundes liegt in der Regel eine nicht-sequentielle, nebenläufige Arbeitsweise zugrunde. Für die Modellierung derartiger Geräte und Gesamtsysteme eignen sich insbesondere Petrinetze, zu deren wesentlichen Merkmalen die Abbildbarkeit der Nebenläufigkeit und die Möglichkeit der Synchronisation von Abläufen zählen. Die Petrinetztheorie geht auf die in den 60er Jahren von Carl Adam Petri verfasste Dissertation „Kommunikation mit Automaten" zurück. Sie stellt eine Weiterentwicklung der Automatentheorie dar.

Nachfolgend wird zunächst auf die Definition von Automaten und die mit ihnen verbundenen Modellierungsrestriktionen eingegangen. Es schließt sich eine detaillierte Diskussion zur Petrinetztheorie, deren konzeptuelle Grundgedanken in die entwickelte Modellierungssprache DevXML eingeflossen sind, an. Die für das Verständnis der in Hauptkapitel 4 vorgestellten Verfahren zur Nutzerführung (vgl. Kapitel 4.3.3 „Navigation") sowie Systemselbstüberwachung (vgl. Kapitel 4.3.4 „Modellzuverlässigkeit") notwendigen Kenntnisse werden in den aufeinander aufbauenden Unterkapiteln „Grundlegende Eigenschaften", „Dynamisches Verhalten" und „Invariantenkalkül" vermittelt.

Ein deterministischer endlicher Automat oder eine endliche sequentielle Maschine (finite sequential machine) ist ein 5-Tupel

$$A = (\Sigma, Z, \delta, s_0, E) \text{ mit} \tag{3.1}$$

- einem endlichen, nichtleeren Eingabealphabet Σ,
- einer endlichen, nichtleeren Menge Z von Zuständen,
- einer Übergangsfunktion δ: $Z \times \Sigma \rightarrow Z$,
- einem Startzustand $s_0 \in Z$ und
- einer Menge von Endzuständen $E \subseteq Z$ (akzeptierenden Zuständen)[122].

Für die Überführung von A aus dem Zustand s in den Zustand s' durch Eingabe des Buchstabens a wird die Schreibweise

$$\delta(s, a) = s' \tag{3.2}$$

gewählt.

[122] Erk/Priese: Theoretische Informatik, S. 63 ff.

Die Mehrfachanwendung der Übergangsfunktion wird im Weiteren mit δ^* gekennzeichnet. Die induktive Definition von δ^* lautet:

$$\delta^*(s,\varepsilon) = s \tag{3.3}$$

$$\delta^*(s,wa) = \delta(\delta^*(s,w),a) \tag{3.4}$$

ε stellt hierbei die leere Eingabefolge dar. Gilt $\delta^*(s_0,w) \in E$, so wird w akzeptiert. Automaten eignen sich für die Modellierung technischer Systeme. Eine häufige angewendete Darstellungsform von Automaten ist der Graph. Kanten werden mit Elementen aus Σ und Knoten mit Elementen aus Z beschriftet. Führt eine Kante mit der Beschriftung a von s_x nach s_y so ist dies äquivalent zu $\delta(s_x,a) = s_y$. Der Startknoten s_0 wird mit einem Pfeil annotiert, finale Zustände sind durch einen Doppelkreis gekennzeichnet.

Die Anwendung sei exemplarisch anhand eines Getränkeautomaten verdeutlicht. Ein Getränk kostet 1 € und es werden 50 ct- sowie 1 €-Münzen angenommen. Die Geldrückgabe ist jederzeit möglich. Es gelte:

- Σ = {50 ct, 1 €, Rückgabe, Getränk entnehmen}
- Z = {s_0, s_1, s_2}
- $\delta(s_0,50\ ct) = s_1$, $\delta(s_0,1\ €) = s_2$, $\delta(s_1,50\ ct) = s_2$, $\delta(s_1,Rückgabe) = s_0$, $\delta(s_2,Rückgabe) = \delta(s_2,Getränk\ entnehmen) = s_0$
- s_0 ist Startzustand
- E = {s_0}

Ein akzeptiertes Wort ist beispielsweise w = {50 ct, 50 ct, Getränk entnehmen}, da $\delta^*(s_0, w) = s_0 \in E$. Die zugehörige graphische Repräsentation zeigt Abbildung 47.

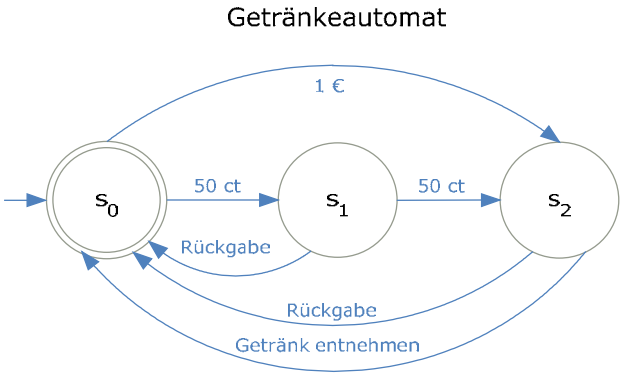


Abbildung 47 Ein endlicher deterministischer Automat. Ein Getränk kostet 1 €. Es werden sowohl 50 ct- als auch 1 €-Münzen akzeptiert. Die Geldrückgabe ist in den Zuständen s_1 und s_2 möglich.

Wie oben bereits erwähnt laufen in Systemen Vorgänge häufig parallel ab. Mit Automaten lässt sich Parallelität jedoch nicht darstellen. Petrinetze, die eine Obermenge von Automaten bilden[123], unterliegen dieser Einschränkung nicht. Aufgrund dessen fokussieren sich die weiteren Betrachtungen auf die Petrinetztheorie.

[123] Ein Petrinetz, bei dem jede Transition höchstens einen Vor- und einen Folgeplatz hat, entspricht einem endlichen deterministischen Automat (vgl. KÖNIG/QUÄCK: Petri-Netze in der Steuerungs- und Digitaltechnik, S. 36)

In dem vorliegenden Kapitel wird konsequent die Notation der Relationenalgebra verwendet, um auf dieser formalisierten Grundlage den im weiteren Verlauf benötigten Zugang zu algorithmisierbaren Darstellungsformen und zu mathematischen Analyseverfahren zu erhalten.

3.1 Grundlegende Eigenschaften

Ein Petrinetz besteht aus den vier Netzelementen Platz, Marke, Transition und Kante. Die zugehörigen Symbole sind Tabelle 19 zu entnehmen. Ein wesentlicher Vorteil der graphischen gegenüber der textuellen Notation besteht in der intuitiven Erfassbarkeit.

Bezeichnung	Graphische Darstellung
Platz (Synonyme: Stelle, Bedingung)	
Marke (Synonym: Token)	
Transition (Synonym: Übergang)	
Kante (Pre- und Postkanten) mit Kantengewicht W	

Tabelle 19 Petrinetze setzen sich aus den Netzelementen Platz, Marke, Transition und Kante zusammen. Für deren graphische Darstellung werden rechtsseitig aufgeführte Symbole verwendet.

Im Rahmen dieser Arbeit wird ein Platz als ein Geräte- bzw. Systemzustand aufgefasst. Die aktuelle Zustandsausprägung erfolgt über die einem Platz zugeordnete Anzahl von Marken. Für die Modellierung von Zustandsübergängen werden Transitionen verwendet, die in Kombination mit gerichteten Kanten die Abhängigkeiten zwischen Geräte- bzw. Systemzuständen abbilden. Anhand des in Abbildung 48 dargestellten Binärschalters, der die Zustände „Geschlossen“ und „Geöffnet“ annehmen kann, wird exemplarisch die Verwendung der vorgenannten Netzelemente demonstriert.

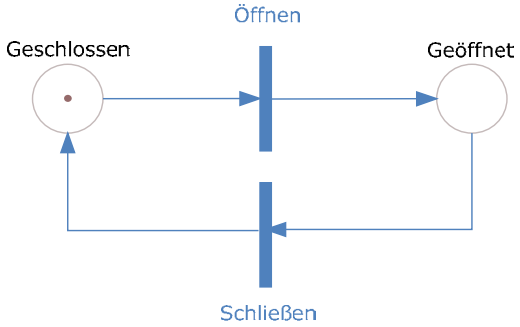


Abbildung 48 Petrinetz eines Binärschalters. Die Zustände „Geschlossen“ und „Geöffnet“ werden als Plätze, die durchführbaren Handlungen „Öffnen“ und „Schließen“ als Transitionen modelliert.

Jedes Petrinetz ist eindeutig über seine Plätze, Transitionen, Prekanten, Postkanten, Kantengewichte und Initialmarkierung definiert. Die relationenalgebraische Formulierung führt zu Definition 1.

Definition 1
Ein Petrinetz N ist definiert als Tupel

$$N = (P, T, Pre, Post, W, M_0) \text{ mit} \tag{3.5}$$

1. Menge der Plätze $P = \{p_1, p_2, ..., p_n\}$ ist endlich und nichtleer

2. Menge der Transitionen $T = \{t_1, t_2, ..., t_m\}$ ist endlich und nichtleer

3. $P \cap T = \emptyset$

4. $Pre \subseteq P \times T$

5. $Post \subseteq T \times P$

6. *Kantengewicht* $W : Pre \cup Post \rightarrow \mathbb{N}$

7. *Initialmarkierung* $M_0 : P \rightarrow \mathbb{N}$

Petrinetze gehören zur Klasse der bipartiten Graphen. Ein bipartiter Graph $G = (V \cup W, E)$ ist dadurch gekennzeichnet, dass die Knotenmenge aus zwei disjunkten Teilmengen V und W besteht und alle Kanten E jeweils ein Element aus V und W als Start- bzw. Endknoten besitzen. Bei Petrinetzen setzt sich die Knotenmenge K aus den Teilmengen P und T zusammen. Siehe hierzu Definition 2. Wahlweise stellen Transitionen oder Plätze Schnittstellen für externe Ereignisse dar. Infolge der Außeneinwirkung wird im ersten Fall Einfluss auf das Feuern einer Transition, im zweiten Fall Einfluss auf die Belegung mit Marken genommen[124].

Definition 2
Die Vereinigung der Menge der Plätze P mit der Menge der Transitionen T heißt Knotenmenge K:

$$K = P \cup T \tag{3.6}$$

Der Begriff der Flussrelation gemäß Definition 3 wird eingeführt, um die Gesamtheit aller Pre- und Postkanten zu benennen.

Definition 3
Die Vereinigung der Menge der Prekanten Pre mit der Menge der Postkanten $Post$ heißt Flussrelation F:

$$F = Pre \cup Post \subseteq (P \times T) \cup (T \times P) \tag{3.7}$$

[124] Götze: Dialogmodellierung für multimediale Benutzerschnittstellen, S. 53.

Das Petrinetz N stellt sich in diesem Fall als 5-Tupel $N = (P, T, F, W, M_0)$ dar. Für die Bezugnahme auf die Gesamtheit aller vor- respektive nachfolgenden Knotenelemente eines Platzes oder einer Transition werden die Begriffe Menge der Eingangsplätze, Menge der Eingangstransitionen, Vorbereich, Menge der Ausgangsplätze, Menge der Ausgangstransitionen und Nachbereich eingeführt. Siehe hierzu Definition 4 bis Definition 9.

Definition 4
Die Menge aller Plätze, von denen eine Kante zur Transition t führt, heißt Menge der Eingangplätze In(t):

$$In(t) = \{p \mid (p,t) \in Pre\} \tag{3.8}$$

Definition 5
Die Menge aller Transitionen, von denen eine Kante zum Platz p führt, heißt Menge der Eingangstransitionen In(p):

$$In(p) = \{t \mid (t,p) \in Post\} \tag{3.9}$$

Die Definition 4 und Definition 5 werden auch häufig unter Bezugnahme auf die Flussrelation F zusammengefasst. Hierzu wird der Begriff des Vorbereiches eingeführt.

Definition 6
Für den Vorbereich von x, der mit $\cdot x$ bezeichnet wird, gilt:

$$\bullet x = \{y \mid (y,x) \in F\} = In(x) = \{y \mid (y,x) \in Pre \vee (y,x) \in Post\} \tag{3.10}$$

Definition 7
Die Menge aller Plätze, zu denen eine Kante von der Transition t führt, heißt Menge der Ausgangsplätze Out(t):

$$Out(t) = \{p \mid (t,p) \in Post\} \tag{3.11}$$

Definition 8
Die Menge aller Transitionen, zu denen eine Kante von dem Platz p führt, heißt Menge der Ausgangstransitionen Out(p):

$$Out(p) = \{t \mid (p,t) \in Pre\} \tag{3.12}$$

Analog zur Definition des Vorbereichs von x wird der Nachbereich von x definiert.

Definition 9
Für den Nachbereich von x, der mit $x\cdot$ bezeichnet wird, gilt:

$$x\bullet = \{y \mid (x,y) \in F\} = Out(x) \tag{3.13}$$

Für die nähere Beschreibung von Knoten werden häufig Verzweigungseigenschaften mit angegeben. Die Begriffe rückwärts- und vorwärtsverzweigt sind wie folgt definiert.

Definition 10
Ein Knoten x, dessen Vorbereich mehr als 1 Element umfasst, heißt rückwärtsverzweigt:

$$|\bullet x| > 1 \tag{3.14}$$

Definition 11
Ein Knoten x, dessen Nachbereich mehr als 1 Element umfasst, heißt vorwärtsverzweigt:

$$|x\bullet| > 1 \tag{3.15}$$

Der Zustand eines Petrinetzes wird über die Netzmarkierung M beschrieben. Essentiell für die Zustandskodierung ist daher Definition 12.

Definition 12
Eine Markierung M ist eine mathematische Vorschrift, die jedem Platz des Petrinetzes eine natürliche Zahl zuordnet:

$$M : P \to \mathbb{N} \tag{3.16}$$

Die einem Platz p_i zugeordnete Zahl $M(p_i)$ gibt hierbei die Anzahl der Marken auf diesem Platz an. Der zum Petrinetz gehörige Zustandsraum ist $\mathbb{N}^p$. Eine spezielle Klasse stellen binäre Petrinetze dar. Jeder Platz beinhaltet maximal eine Marke. Die zugehörige Abbildung lautet demnach:

$$M : P \to \{0,1\} \tag{3.17}$$

Für viele Systemmodelle ist diese Form der Abbildung vollkommen ausreichend. Markierungen werden auch als Spaltenvektoren geschrieben. Für die Markierung $M \in \mathbb{N}^p$ gilt dann:

$$M = \begin{pmatrix} M(p_1) \\ M(p_2) \\ \dots \\ M(p_n) \end{pmatrix} \tag{3.18}$$

Für die Taxonomie von Petrinetzen werden häufig noch weitere grundlegende Eigenschaften herangezogen. Mit der Bildung von Subnetzklassen, die gewissen Einschränkungen unterliegen, sind dann erweiterte Aussagen möglich. Typische Subnetzklassen sind z. B. Synchronisationsgraphen, Zustandsmaschinen, Free-Choise-Netze und einfache Netze[125]. An dieser Stelle wird hierauf nicht näher eingegangen, da der im Rahmen dieser Arbeit vorgestellte netztheoretische Ansatz für die Geräte- und Systemmodellierung grundsätzlich alle Formen von binären Petrinetzen zulässt.

[125] König/Quäck: Petri-Netze in der Steuerungs- und Digitaltechnik, S. 35 ff.

3.2 Dynamisches Verhalten

Zustandswechsel von Petrinetzen werden durch Markenflüsse modelliert. Kantengewichte werden für die quantitative Bestimmung der Markenflüsse eingeführt. Sie haben sowohl Einfluss auf die Feuerbarkeit einer Transition als auch auf jede sich durch Feuern ergebende Folgemarkierung. Siehe hierzu Definition 13 bis Definition 15.

Definition 13
Ein Kantengewicht W gibt die Anzahl der beim Feuern über die Pre- und Postkanten fließenden Marken an. Es gilt folgende Abbildungsvorschrift:

$$W : Pre \cup Post = F \rightarrow \mathbb{N} \tag{3.19}$$

In Graphen ist die Angabe des Kantengewichtes an dem jeweiligen Netzelement vorzunehmen. Beträgt das Kantengewicht 1, kann die Angabe unterbleiben.

Definition 14
Eine Transition t heißt genau dann feuerbar, wenn jeder ihrer Vorplätze p_i mindestens $W(p_i,t)$ Marken beinhaltet. O.B.d.A. gilt folgende Transitionsregel:

$$In(t) \subseteq \{p \mid M(p) \geq W(p,t)\} (\text{äquivalent zu:} \forall p \in \bullet t : M(p) \geq W(p,t)) \tag{3.20}$$

Bei binären Petrinetzen heißt eine Transition genau dann feuerbar, wenn gilt:

$$In(t) \subseteq \{p \mid M(p) = 1\} (\text{äquivalent zu:} \forall p \in \bullet t : M(p) = 1) \tag{3.21}$$

Definition 15
Ausgehend von der Markierung M wird durch Feuern der Transition t das Petrinetz in die Folgemarkierung M' überführt. Man schreibt auch $M[t> M'$. M' ist hierbei wie folgt definiert:

$$M'(p) = \begin{cases} M(p) - W(p,t) \text{ wenn } p \in In(t) \backslash Out(t) \text{ (äquivalent zu:} p \in \bullet t \wedge p \notin t \bullet) \\ M(p) + W(t,p) \text{ wenn } p \in Out(t) \backslash In(t) \text{ (äquivalent zu:} p \in t \bullet \wedge p \notin \bullet t) \\ M(p) - W(p,t) + W(t,p) \text{ wenn } p \in In(t) \cap Out(t) \text{ (äquivalent zu: } p \in \bullet t \wedge p \in t \bullet) \\ M(p) \text{ sonst} \end{cases} \tag{3.22}$$

Bei binären Petrinetzen gilt analog:

$$M'(p) = \begin{cases} 0 \text{ wenn } p \in In(t) \backslash Out(t) \text{ (äquivalent zu:} p \in \bullet t \wedge p \notin t \bullet) \\ 1 \text{ wenn } p \in Out(t) \text{ (äquivalent zu:} p \in t \bullet) \\ M(p) \text{ sonst} \end{cases} \tag{3.23}$$

Im Folgenden werden die Begriffe Erreichbarkeitsmenge, k-beschränkt, reproduzierbar und lebendig definiert, um so die mathematische Grundlage für die Beschreibung des dynamischen Verhaltens von Petrinetzen in Hinsicht auf erreichbare Markierungen, die maximale Anzahl von Marken pro Platz und die Reproduzierbarkeit von Markierungen zu schaffen.

Definition 16
Die Menge der Markierungen, die durch anwendbare Feuersequenzen aus dem Initialzustand M_0 erreicht werden können, heißt Erreichbarkeitsmenge $E(M_0)$. $E(M_0)$ ist somit die kleinste Menge von Markierungen, die nachfolgende Eigenschaften erfüllt:

1. $M_0 \in E(M_0)$ (3.24)
2. $In(t) \subseteq \{p \mid M(p) = 1\}\ (M_1[t > M_2 \wedge M_1 \in E(M_0)) \Rightarrow M_2 \in E(M_0)$ (3.25)

Definition 17
Ein Petrinetz N heißt beschränkt, wenn eine natürliche Zahl k existiert, sodass jede aus der Initialmarkierung M_0 erreichbare Markierung M die Eigenschaft besitzt, dass kein Platz die Markenzahl k überschreitet. Das Petrinetz N wird dann auch als k-beschränkt bezeichnet. Formal gilt:

$$N \text{ heißt } (k\text{-})\text{beschränkt} \Leftrightarrow \exists k \in \mathbb{N} : \forall M \in E(M_0) : \forall p \in P : k \geq M(p) \quad (3.26)$$

Jedes binäre Petrinetz N ist per Definition 1-beschränkt. Da jeder Platz p_i mit $i = 1...n$ somit nicht mehr als zwei Zustände einnehmen kann, sind für N maximal 2^n unterschiedliche Markierungen möglich.

Definition 18
Eine Markierung M heißt reproduzierbar, wenn eine Transition t existiert, die das Netz aus der Markierung M in die Markierung M_1 überführt und wenn die Erreichbarkeitsmenge $E(M_1)$ die Markierung M einschließt:

$$\exists t \in T : M[t > M1 \wedge M \in E(M1) \quad (3.27)$$

Definition 19
Eine Transition t heißt lebendig, wenn für jede Markierung M, die aus M_0 erreicht werden kann, eine Markierung M' existiert, die von M erreichbar ist und aus der t gefeuert werden kann:

$$t \text{ ist lebendig} \Leftrightarrow \forall M \in \mathrm{E}(M_0) : \exists M' \in \mathrm{E}(M), \text{sodass } t \text{ aus } M' \text{ feuerbar ist} \quad (3.28)$$

Definition 20
Eine Petrinetz N ist genau dann lebendig, wenn alle Transitionen von N lebendig sind.

$$N \text{ ist lebendig} \Leftrightarrow \forall t \in T : t \text{ ist lebendig} \quad (3.29)$$

In einem lebendigen Petrinetz sind nicht notwendigerweise alle Markierungen reproduzierbar. Dies zeigt Abbildung 49. Nachdem das Netz aus der Markierung $M = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ durch Feuern der Transition t_2 in die Markierung $M' = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ übergegangen ist, kann die Markierung M nicht wieder erreicht werden. Gleichwohl sind die Transitionen t_1 und t_2 lebendig – und somit auch das Petrinetz.

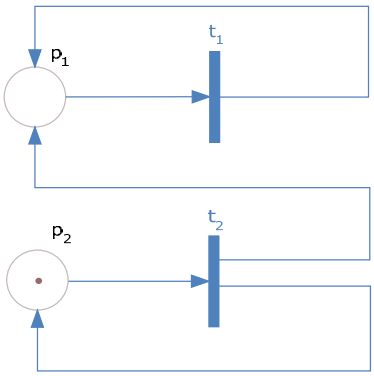


Abbildung 49 Lebendiges Netz mit einer nicht reproduzierbaren Markierung $M = (0\ 1)^T$. Wird die Transition t_2 aus M gefeuert, so geht das Netz in $M' = (1\ 1)^T$ über. Die Markierung M kann aus M' nicht wieder hergestellt werden.

3.3 Invariantenkalkül

Invarianten geben unveränderliche Eigenschaften wieder. In der Petrinetztheorie wird zwischen T- und P-Invarianten unterschieden. Die Bezeichnung T-Invariante geht auf die Invarianz einer Markierung gegenüber einer Feuersequenz, die aus einer Folge von Transitionen besteht, zurück. Die Bezeichnung P-Invariante ist auf die Konstanz der Summe platzspezifisch gewichteter Marken zurückzuführen. Das Invariantenkalkül basiert auf der Matrixdarstellung von Markenflüssen. Die Kodierung markenflussrelevanter Informationen erfolgt hierbei über sogenannte Inzidenzmatrizen.

Definition 21
Die Inzidenzmatrix I des Netzes N ist bestimmt durch:

$$I = \begin{array}{c} \begin{matrix} t_1 & & t_m \end{matrix} \\ \begin{pmatrix} t_{11} & \cdots & t_{m1} \\ \vdots & \ddots & \vdots \\ t_{1n} & \cdots & t_{mn} \end{pmatrix} \end{array} \begin{matrix} p_1 \\ \\ p_n \end{matrix} \tag{3.30}$$

mit

$$t_{ij} = \begin{cases} +W(i,j) \text{ wenn } p_j \in Out(t_i) \backslash In(t_i) \\ -W(j,i) \text{ wenn } p_j \in Pre(t_i) \backslash Out(t_i) \\ +W(i,j) - W(j,i) \text{ wenn } p_j \in Out(t_i) \cap In(t_i) \\ 0 \text{ sonst} \end{cases}$$

Der Eintrag t_{ij} der Inzidenzmatrix I gibt also an, um welchen Wert sich die Markenzahl des Platzes p_j bei Feuern der Transition t_i ändert. Für das obige Beispielnetz ergibt sie sich zu:

$$I = \begin{array}{c} \begin{matrix} t_1 & t_2 & t_3 \end{matrix} \\ \begin{pmatrix} -2 & 0 & 2 \\ -1 & 0 & 1 \\ 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \end{matrix} \tag{3.31}$$

Werden nur schlingenfreie Petrinetz zugelassen, so ist die Inzidenzmatrix umkehrbar eindeutig. Andernfalls gehen durch die Inzidenzmatrix strukturelle Informationen verloren. Siehe hierzu auch Abbildung 50, die zwei strukturell verschiedenartige Petrinetze mit identischer Inzidenzmatrix

$$I = \begin{array}{c} \begin{matrix} t_1 & t_2 \end{matrix} \\ \begin{pmatrix} 0 & -1 \\ 0 & 1 \\ 1 & 0 \end{pmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \end{matrix} \tag{3.32}$$

zeigt. Durch Feuern der Transition t_1 ändert sich bei beiden Petrinetzen die jeweils resultierende Markenbelegung von p_1 nicht. Im links abgebildeten Petrinetz heben sich Markenzufluss und -abfluss gegenseitig auf. Bei dem rechts abgebildeten Petrinetz hingegen sind p_1 und t_1 weder über Pre- noch Postkanten miteinander verbunden.

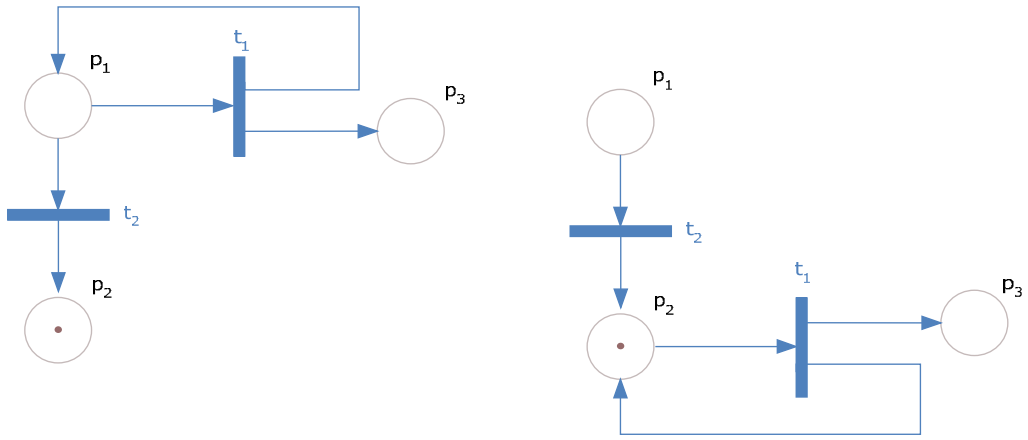


Abbildung 50 Sofern Petrinetze mit Schlingen zugelassen werden, sind deren Inzidenzmatrizen nicht umkehrbar eindeutig. Zu sehen sind strukturell verschiedenartige Petrinetze mit identischer Inzidenzmatrix

Existiert eine aus der Markierung M heraus anwendbare Schaltfolge, die d_1-mal die Transition t_1, d_2-mal die Transition t_2,..., d_m-mal die Transition t_m beinhaltet, so geht das Petrinetz in die Markierung M^* über und es gilt:

$$M^* = M + I\phi \tag{3.33}$$

mit

$$\phi = (d_1 \quad d_2 \quad \ldots \quad d_m)^T$$

Der Häufigkeitsvektor ϕ wird auch als Parikh-Vektor bezeichnet. Für eine Schaltsequenz, die die Markierung M in sich selbst überführt, gilt:

$$M = M + I\phi \Leftrightarrow I\phi = 0 \tag{3.34}$$

Diese Gleichung ist Ausgangspunkt für die Definition des Begriffes der T-Invariante. Es ist hierbei zu beachten, dass die Gleichung gegebenenfalls auch von Parikh-Vektoren, denen keine anwendbare Schaltsequenz zugeordnet werden kann, erfüllt wird.

Definition 22
Ein Vektor ϕ_T mit Koeffizienten aus $\mathbb{Z}$ heißt T-Invariante, wenn gilt:

$$I\phi_T = 0 \tag{3.35}$$

Satz 1
Jedes lebendige und beschränkte Petrinetz besitzt mindestens eine T-Invariante mit ausschließlich positiven ganzen Koeffizienten (ungleich Null).

Beweis
Wegen der Beschränktheit des Netzes N ist seine Erreichbarkeitsmenge endlich. Sei t_m nun eine beliebige von n Transitionen. Aufgrund der Lebendigkeit muss dann eine Feuersequenz $\phi_{T,m}$ existieren, die t_m beinhaltet und hierbei zum Durchlaufen derselben Netzmarkierungen führt. Eine T-Invariante mit ausschließlich positiven ganzen Koeffizienten erhält man durch Linearkombination aller $\phi_{T,k}$ mit k=1...n.

Satz 2
Die Umkehraussage von Satz 1 gilt im Allgemeinen nicht. Die Existenz einer T-Invariante mit ausschließlich positiven ganzen Koeffizienten (ungleich Null) ist notwendige, nicht aber hinreichende Bedingung für ein lebendiges und beschränktes Netz.

Beweis
Ein Petrinetz, das Satz 2 belegt, ist in Abbildung 51 dargestellt. Jeder Vektor $\phi_T \in \mathbb{Z}^1$ stellt für das gegebene Petrinetz N offensichtlich eine T-Invariante dar. Ein Vektor mit ausschließlich positiven ganzen Koeffizienten ist somit z. B. der Vektor $\phi_T = (3)$. Es ist ersichtlich, dass aus der Markierung $M = (0 \quad 1)^T$ keine Markierung erreicht werden kann, in der die Transition t_1 feuerbar ist. Somit ist die Eigenschaft der Lebendigkeit für t_1 und in der Folge für das Petrinetz N nicht erfüllt.

Hingegen führt die abweichende Initialmarkierung $\overline{M} = (1 \quad 1)^T$ zu einem lebendigen Netz, für das unter Berücksichtigung des Parikh-Vektors eine Schaltfolge existiert – hier: dreimaliges Feuern der Transition t_1 –, die die Markierung $\overline{M}$ reproduziert.

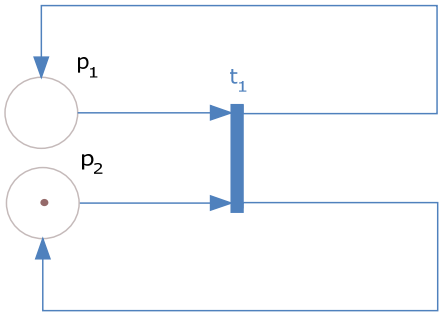


Abbildung 51 Nicht-lebendiges Netz. Aus der Markierung $M = (0\ 1)^T$ kann die Transition t_1 niemals gefeuert werden.

Die Berechnung platzspezifisch gewichteter Markensummen unter Verwendung von P-Invarianten ermöglicht die Prüfung eines Geräte- bzw. Systemzustandes auf Validität. In der Literatur wird häufig auch die synonyme Bezeichnung S-Invariante verwendet. Die Präfixe P- und S- stehen hierbei für Platz bzw. Stelle. Im Rahmen dieser Arbeit wird durchgängig die Bezeichnung P-Invariante genutzt.

Definition 23
Ein Vektor ϕ_P mit Koeffizienten aus $\mathbb{Z}$ heißt P-Invariante, wenn gilt:

$$I^T \phi_P = 0 \tag{3.36}$$

Satz 3
Gilt für eine P-Invariante ϕ_P die Beziehung $M^T \phi_P = c$, so folgt für eine Markierung M^* mit $M^{*T} \phi_P \neq c$, dass sie von M aus nicht erreichbar ist.

Beweis
Folgende Umformungen weisen die Konstanz der Markensumme eines Petrinetzes, für das eine P-Invariante ϕ_P existiert, nach:

$$\begin{aligned}
& M^* = M + I\phi \\
\Leftrightarrow \quad & M^* - M = I\phi \\
\Leftrightarrow \quad & (M^* - M)^T = (I\phi)^T && \text{es gilt: } (AB)^T = B^T A^T \\
\Leftrightarrow \quad & M^{*T} - M^T = \phi^T I^T && \\
\Leftrightarrow \quad & (M^{*T} - M^T)\phi_P = \phi^T I^T \phi_P && \text{gemäß Definition gilt: } I^T \varphi_P = 0 \\
\Leftrightarrow \quad & (M^{*T} - M^T)\phi_P = 0 && \\
\Leftrightarrow \quad & M^{*T} \phi_P = M^T \phi_P &&
\end{aligned} \tag{3.37}$$

Der Zusammenhang zwischen der Beschränktheit eines Petrinetzes und der Existenz einer P-Invariante mit ausschließlich positiven ganzen Koeffizienten (ungleich Null) wird in Satz 4 und Satz 5 beschrieben.

Satz 4
Jedes Petrinetz *N* mit einer P-Invariante, die ausschließlich positive ganze Koeffizienten (ungleich Null) beinhaltet, ist beschränkt.

Beweis
Sei ϕ_P eine P-Invariante, die ausschließlich positive ganze Koeffizienten beinhalte. Dann gilt nach Satz 3 für alle erreichbaren Markierungen $M_1,\ldots, M_n$ der Zusammenhang $M_1^T\phi_P = M_2^T\phi_P = \ldots = M_n^T\phi_P = k$. Da alle Elemente von $M_1 - M_n$ positiv sind, ist N offensichtlich k-beschränkt.

Satz 5
Die Umkehraussage von Satz 4 gilt im Allgemeinen nicht. Sie gilt auch dann nicht, wenn das Petrinetz lebendig ist. Ist ein Petrinetz beschränkt (und gegebenenfalls lebendig), so folgt daraus also nicht, dass es eine P-Invariante mit ausschließlich positiven ganzen Koeffizienten (ungleich Null) besitzt.

Beweis
Das Petrinetz in Abbildung 52 ist ein Beleg für Satz 5. Es ist beschränkt und besitzt ausschließlich die triviale P-Invariante $\phi_P = 0$[126].

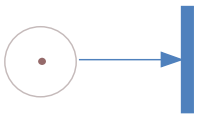


Abbildung 52 Jedes Petrinetz, das eine P-Invariante mit ausnahmslos positiven ganzen Koeffizienten besitzt, ist beschränkt. Die Umkehraussage gilt im Allgemeinen nicht. Abgebildet ist ein beschränktes Petrinetz mit ausschließlich trivialer P-Invariante.

3.4 Beispiel

Anhand des in Abbildung 53 dargestellten Petrinetzes werden die grundlegenden Eigenschaften, die Regeln des Markenflusses, das dynamische Verhalten und das Invariantenkalkül exemplarisch beschrieben.

[126] siehe auch Reisig: Petrinetze, S. 94.

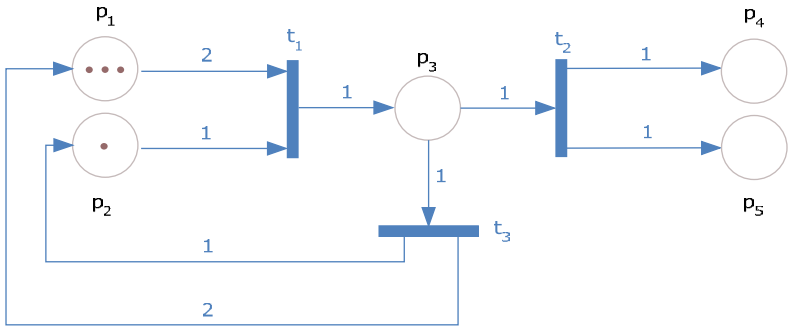


Abbildung 53 Petrinetz in der Markierung $M = (3\ 1\ 0\ 0\ 0)^T$. Auf dessen Grundlage wird ein konkreter Anwendungsbezug zur vorstehenden Petrinetztheorie hergestellt.

Wie oben ausgeführt umfassen die grundlegenden Eigenschaften die Netzelemente von P sowie die Vor- und Nachbereiche. Für das Petrinetz $N = (P, T, Pre, Post, W, M_0)$ gilt:

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$
$$T = \{t_1, t_2, t_3\}$$
$$Pre = \{(p_1,t_1),(p_2,t_1),(p_3,t_2),(p_3,t_3)\}$$
$$Post = \{(t_1,p_3),(t_2,p_4),(t_2,p_5),(t_3,p_1),(t_3,p_2)\}$$
$$\text{(bzw. } F = Pre \cup Post = \{(p_1,t_1),(p_2,t_1),(p_3,t_2),(p_3,t_3),(t_1,p_3),(t_2,p_4),(t_2,p_5),(t_3,p_1),(t_3,p_2)\}\text{)}$$
$$W(p_1,t_1) = W(t_3,p_1) = 2$$
$$W(p_2,t_1) = W(t_1,p_3) = W(p_3,t_2) = W(t_2,p_4) = W(t_2,p_5) = W(p_3,t_3) = W(t_3,p_2) = 1$$
$$M_0 = (3\ \ 1\ \ 0\ \ 0\ \ 0)^T$$

Die Vor- und Nachbereiche bestimmen sich zu:

$$In(t_1) = {\bullet t_1} = \{p_1, p_2\} \qquad Out(t_1) = t_1{\bullet} = \{p_3\}$$
$$In(t_2) = {\bullet t_2} = \{p_3\} \qquad Out(t_2) = t_2{\bullet} = \{p_4\}$$
$$In(t_3) = {\bullet t_3} = \{p_3\} \qquad Out(t_3) = t_3{\bullet} = \{p_1, p_2\}$$
$$In(p_1) = {\bullet p_1} = \{t_3\} \qquad Out(p_1) = p_1{\bullet} = \{t_1\}$$
$$In(p_2) = {\bullet p_2} = \{t_3\} \qquad Out(p_2) = p_2{\bullet} = \{t_1\}$$
$$In(p_3) = {\bullet p_3} = \{t_1\} \qquad Out(p_3) = p_3{\bullet} = \{t_2, t_3\}$$
$$In(p_4) = {\bullet p_4} = \{t_2\} \qquad Out(p_4) = p_4{\bullet} = \varnothing$$
$$In(p_5) = {\bullet p_5} = \{t_2\} \qquad Out(p_5) = p_5{\bullet} = \varnothing$$

Der Markenfluss wird anhand von Abbildung 54 veranschaulicht. In der Markierung M_0 ist gemäß Definition 14 ausschließlich die Transition t_1 feuerbar. Die Überführung der Markierung M_0 in die Folgemarkierung M' soll beispielhaft an dem Feuern der Transition t_1 veranschaulicht werden. Gemäß Definition 15 resultiert durch den Markenfluss die Folgemarkierung $M' = (1\ \ 0\ \ 1\ \ 0\ \ 0)^T$.

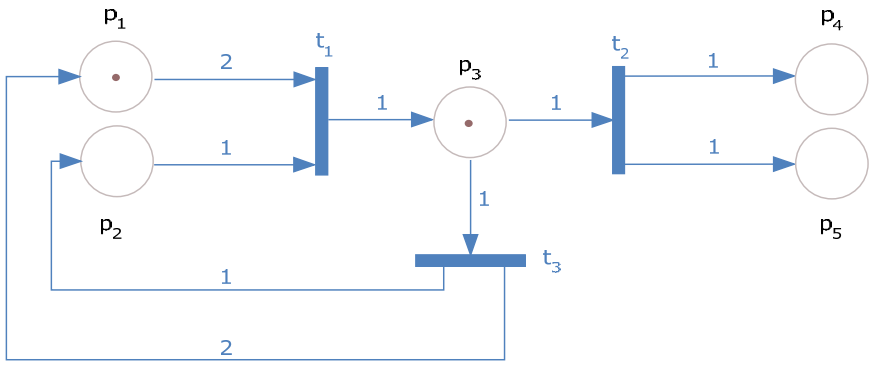


Abbildung 54 Petrinetz in der Markierung $M' = (1\ 0\ 1\ 0\ 0)^T$. Der Übergang von $M_0 = (3\ 1\ 0\ 0\ 0)^T$ in M' wird durch Feuern der Transition t_1 bewirkt.

Auf das dynamische Verhalten von P wird nachfolgend eingegangen. Ausgehend von der Initialmarkierung $M_0 = (3\ \ 1\ \ 0\ \ 0\ \ 0)^T$ können in dem Petrinetz insgesamt drei verschiedene Markierungen erreicht werden. Die Erreichbarkeitsmenge $E(M_0)$ ergibt sich zu:

$$E(M_0) = \left\{ \begin{pmatrix} 3 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} \right\} \tag{3.38}$$

Im gegebenen Fall ist die Erreichbarkeitsmenge intuitiv leicht ableitbar. In Kapitel 4 wird ein Algorithmus vorgestellt, der die Erreichbarkeitsmenge eines beliebigen, beschränkten Petrinetzes generiert. Das vorliegende Petrinetz ist 3-beschränkt, da kein Vektor aus $E(M_0)$ einen Koeffizienten mit einem Wert größer 3 beinhaltet. Reproduzierbar sind die Markierungen $M_0 = (3\ \ 1\ \ 0\ \ 0\ \ 0)^T$ und $M_1 = (1\ \ 0\ \ 1\ \ 0\ \ 0)^T$, denn es gilt:

$$M_0 [t_1 > M_1 [t_3 > M_0 \text{ sowie } M_1 [t_3 > M_0 [t_1 > M_1 \tag{3.39}$$

$M_2 = (1\ \ 0\ \ 0\ \ 1\ \ 1)^T$ ist hingegen nicht reproduzierbar und es liegt ferner ein nichtlebendiges Petrinetz vor, da in der Markierung M_2 keine Transition feuerbar ist. Vergleiche hierzu Definition 18 und Definition 20. Die Anwendung des Invariantenkalküls auf das Beispielnetz wird im Folgenden beschrieben. Für die Bestimmung von T-Invarianten ist das Gleichungssystem

$$I\Theta_T = \begin{pmatrix} -2 & 0 & 2 \\ -1 & 0 & 1 \\ 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix} \Theta_T = 0 \tag{3.40}$$

zu lösen.

Für Θ_T ergibt sich der Lösungsraum:

$$\Theta_T = i\begin{pmatrix}1\\0\\1\end{pmatrix} \tag{3.41}$$

Der Lösungsraum beinhaltet keine T-Invariante mit ausschließlich positiven ganzen Koeffizienten. Aus Satz 1 folgt daher, dass das Petrinetz nicht lebendig ist. Dies wurde bereits im Abschnitt „Dynamisches Verhalten“ auf alternative Weise gezeigt. Jede schaltbare Transitionsfolge, die t_1 und t_3 jeweils i-mal beinhaltet, reproduziert demnach die aktuelle Netzmarkierung. So wird beispielsweise für $i=2$ M_0 durch die Transitionsfolge t_1, t_3, t_1, t_3 und M_1 durch die Transitionsfolge t_3, t_1, t_3, t_1 reproduziert. Für die Bestimmung von P-Invarianten ist nachfolgendes Gleichungssystem zu lösen.

$$I^T\Theta_P = \begin{pmatrix}-2 & -1 & 1 & 0 & 0\\ 0 & 0 & -1 & 1 & 1\\ 2 & 1 & -1 & 0 & 0\end{pmatrix}\Theta_P = 0 \tag{3.42}$$

Für Θ_P ergibt sich – z. B. mit dem Additionsverfahren – der Lösungsraum des unterbestimmten Gleichungssystems mit den drei reellen Parametern i, j, k:

$$\Theta_P = i\begin{pmatrix}1\\0\\2\\0\\2\end{pmatrix} + j\begin{pmatrix}0\\1\\1\\0\\1\end{pmatrix} + k\begin{pmatrix}0\\1\\1\\1\\0\end{pmatrix} \tag{3.43}$$

Durch Setzen von $i=5$, $j=1$ und $k=3$ erhält man die P-Invariante $\Theta_{P1} = (5 \quad 4 \quad 14 \quad 3 \quad 11)^T$. Diese hat ausschließlich positive ganze Koeffizienten und überdeckt somit das gesamte Petrinetz. So ist sichergestellt, dass alle Plätze bei der Berechnung der gewichteten Markensumme Berücksichtigung finden. Ein algorithmisches Verfahren zur effizienten Bestimmung von gegebenenfalls vorhandenen P-Invarianten mit dieser Eigenschaft für beliebige Petrinetze wird in Kapitel 4.3.4 vorgestellt. Die potentielle Erkennung von nicht erreichbaren Markierungen wird am Beispiel von $M_3 = (1 \quad 2 \quad 0 \quad 0 \quad 3)^T$ gezeigt. Es gilt:

$$\begin{aligned} &M_0{}^T\Theta_{P1} = M_1{}^T\Theta_{P1} = M_2{}^T\Theta_{P1} \neq M_3{}^T\Theta_{P1} \\ \Leftrightarrow\ &(3\ 1\ 0\ 0\ 0)\begin{pmatrix}5\\4\\14\\3\\11\end{pmatrix} = (1\ 0\ 1\ 0\ 0)\begin{pmatrix}5\\4\\14\\3\\11\end{pmatrix} = (1\ 0\ 0\ 1\ 1)\begin{pmatrix}5\\4\\14\\3\\11\end{pmatrix} \neq (1\ 2\ 0\ 0\ 3)\begin{pmatrix}5\\4\\14\\3\\11\end{pmatrix} \\ \Leftrightarrow\ &19 = 19 = 19 \neq 46 \end{aligned} \tag{3.44}$$

Nach Satz 3 folgt $M_3 \notin E(M_0)$ alleine aus der von 19 abweichenden gewichteten Markensumme $M_3{}^T\Theta_{P1} = 46$.

Kapitel 4

Konzepte

Das vorliegende Kapitel widmet sich der Zielstellung, die Autokonfiguration, die Gerätezuordenbarkeit vor Ort, die Geräteüberwachbarkeit, die Inter-Gerätekommunikation und die Automatisierbarkeit von Abläufen in einem Gesamtkonzept zu vereinen. Auf die zu diesem Zweck zunächst vorzunehmende Auswahl von Basistechnologien mit besonderer Eignung wird in Kapitel 4.1 „Analyse“ eingegangen. Die Grundlage hierfür bildet das Kapitel 2 mit Ausführungen zu lokalen Netzen, verteilten Ad-Hoc-Systemen, der Geräteidentifikation sowie Audio- und Video-Streaming. Siehe hierzu Abbildung 55.

Des Weiteren wird aufgezeigt, welche Anforderungen an Beschreibungssprachen zu stellen sind, damit die Vereinbarkeit mit oben genannter Zielstellung gegeben ist. Das integrale Architekturkonzept DevControl mit DevHTML und DevXML als Bindeglied ist Gegenstand des Kapitels 4.2. Kapitel 4.3 führt detailliert in die Beschreibungssprache DevXML ein.

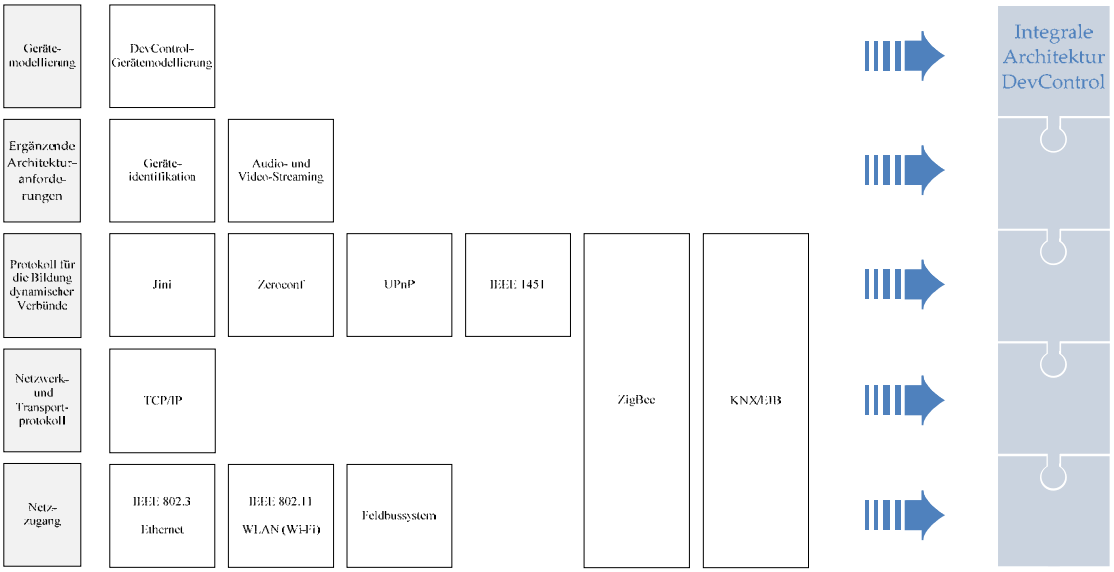


Abbildung 55 Das integrale Architekturkonzept DevControl stützt sich auf Funktionalitäten, die teilweise von Basistechnologien abgedeckt werden. Die Selektion von Basistechnologien mit besonderer Eignung beruht auf einer vergleichenden Analyse.

4.1 Analyse

Das aktuelle Kapitel ist in die Hauptabschnitte „Lokale Netze“, „Verteilte Ad-Hoc-Systeme“, „Geräteidentifikation“, „Audio- und Video-Streaming“ sowie „Gerätemodellierung“ untergliedert. Den Abschluss der vergleichenden Technologiegegenüberstellung bildet eine Übersicht über die vorgenommene Auswahl.

Lokale Netze Im Rahmen der Infrastrukturausgestaltung wird auf bewährte Technologien zurückgegriffen. Ethernet sowie WLAN wurden gegenüber den weiteren Varianten lokaler Netze als vorteilhaft identifiziert. Zum einen ist aufgrund der Interoperabilität mit TCP/IP eine direkte Anbindung an das Internet möglich. Zum anderen sind hohe Datenübertragungsraten erzielbar, die bei der drahtgebundenen Kommunikation bis zu 10 Gbit/s und bei der drahtungebundenen Kommunikation bis zu 600 Mbit/s betragen. Hiermit wird nicht nur der Grundstein für AV-Streaming, sondern auch für Geräte mit datenintensivem Kommunikationsverhalten gelegt. Konzeptuell kommen im Übrigen auch Alternativtechnologien wie UMTS oder Powerline, die sich unmittelbar mit TCP/IP kombinieren lassen, in Frage. Stets ist im Vorfeld eine Prüfung auf Eignung, in die kennzeichnende QoS-Parameter einzubeziehen sind, vorzunehmen. Die weiteren Ausführungen beschränken sich auf Ethernet und WLAN.

Eine definierte Übertragungsgeschwindigkeit kann hier aufgrund des stochastischen Zugriffs auf das Übertragungsmedium nicht zugesichert werden. In der Praxis ist dieser theoretische Aspekt jedoch von nachrangiger Bedeutung, da im statistischen Mittel die Latenzen im Bereich weniger Millisekunden liegen. Durch Überdimensionierung des Kommunikationssystems können Verzögerungen infolge von Kollisionen kompensiert werden. Die Zugriffszeiten in der Größenordnung eines 10 Mbit/s Feldbussystems hält ein 1 Gbit/s Ethernet mit einer Wahrscheinlichkeit nahe 1 ein[127]. Besteht im Einzelfall dennoch die Erfordernis, den unwahrscheinlichen Fall von Medienzugriffszeiten außerhalb eines vorgegebenen Toleranzbereiches auszuschließen, bietet sich Industrial Ethernet an. Im Vergleich zu Feldbussystemen erfolgt die Errichtung lokaler Netze auf Basis von Ethernet oder WLAN unter Verwendung von Standardkomponenten aus der Massenfertigung.

Die WPAN Technologien Bluetooth und ZigBee lassen die Anbindung an das Internet zu. Im Fall von ZigBee konterkarieren die erforderlichen Protokollstapelanpassungen jedoch die eigentliche Zielstellung der Optimierung des 802.15.4-Stacks auf ressourcenbeschränkte Geräte. Ferner stehen die niedrigen Datenübertragungsraten von Bluetooth und ZigBee in Höhe von maximal 3 Mbit/s bzw. 0,25 Mbit/s der Realisierung von Streaming entgegen. Für die Verwendung in einer generischen Systemumgebung stellt diese Restriktion ein Ausschlusskriterium dar. Darüber hinaus sind Echtzeitanforderungen mit WPAN auch annäherungsweise nicht erfüllbar. Die geringen Datenübertragungsraten bieten keinen ausreichenden Spielraum, kollisionsbedingte Zeitverzögerungen durch Überdimensionierung auszugleichen. Die niedrigen Infrastrukturkosten von WPAN wiegen die vorgenannten Nachteile nicht auf.

Feldbussysteme sind für den Einsatz in einem lokalen Umfeld konzipiert. Die in IEC 61158 genormten Feldbusse (z. B. Profibus) sind durch ein reduziertes Schichtenmodell, bei dem im Regelfall Netzwerk-, Transport-, Sitzungs- und Darstellungsschicht (OSI-Schichten 3 bis 6) entfallen, charakterisiert. Die Anwendungsschicht setzt hierbei unmittelbar auf der Verbindungsschicht für die Punkt-zu-Punkt-Kommunikation auf.

[127] Felser: Ethernet als Feldbus.

Feldbussysteme wie P-Net, die Multinet-Strukturen unterstützen, implementieren ergänzend die Netzwerk- und Transportschicht. Für die Anbindung von Feldbussystemen an das Internet sind in jedem Fall Gateways erforderlich. Die resultierende Komplexitätssteigerung des Gesamtsystems wirkt sich nachteilig auf die Fehleranfälligkeit aus. Erzielbare Datenübertragungsraten sind in der gleichen Größenordnung wie bei Ethernet. Folglich ist Streaming ohne Weiteres realisierbar. Durch die zur Anwendung kommenden deterministischen Medienzugriffsverfahren ist die Zusicherung von QoS[128] wie z. B. Datenübertragungsrate und Jitter möglich. Aufgrund dieser Eigenschaft sind Feldbussysteme im Besonderen für Anwendungsfälle mit hohen Anforderungen an die Echtzeitfähigkeit geeignet.

Wie weiter oben ausgeführt weist Industrial Ethernet ein ähnliches Eigenschaftsprofil auf. Die vergleichsweise hohen Kosten von Feldbussystemen laufen dem Teilziel der Kostenminimierung zuwider. Insgesamt wiegen die Nachteile weitaus schwerer als die Vorteile. Somit wird im Weiteren von Feldbussystemen abgesehen.

Von KNX/EIB wird ebenfalls abgesehen. Den hohen Kosten für die Infrastruktur stehen keine Vorteile von Belang für die hier verfolgte Zielstellung gegenüber. Die Realisierung KNX-basierter Gebäudeinstallationen erfordert überdies vertiefte Systemkenntnisse. Diese Prämisse ist mit dem Grundgedanken von Plug and Play unvereinbar.

Verteilte Ad-Hoc-Systeme Von den untersuchten Ad-Hoc-Systemen sind UPnP und Zeroconf aus technologischer Sicht als gleichwertig einzustufen. Es wurde nachgewiesen, dass die jeweiligen Kommunikationsprotokolle mit geringem Aufwand in eigene Anwendungen integriert werden können. Bei niedrigem Ressourcenbedarf wird von beiden Technologien ein den Anforderungen entsprechender Funktionsumfang dargeboten.

Ausschlaggebend für die Einbeziehung von UPnP in das finale Konzept ist die Verbreitung im Markt. Die große Vielfalt an UPnP-konformen Produkten wie Routern, Medienservern und Medienrenderern begünstigt die zeitoptimierte Entwicklung hiervon abgeleiteter Geräte. Mit den in Kapitel 2.2.4 vorgestellten TEDS der IEEE 1451-Standardfamilie können Eigenschaften von Aktoren und Sensoren beschrieben werden. Die Informationskodierung wird in Form strukturierter Datenblöcke mit dem Ziel vorgenommen, elektronische Speicherbausteine möglichst klein und kostengünstig zu halten. Infolge der Kodierung auf Bitebene nimmt die Lesbarkeit ab. Die auf dem Chipmarkt vollzogene Marktentwicklung hat dazu geführt, dass mittlerweile Module mit hohen Speicherdichten kostengünstig zu erwerben sind.

Der Endkundenpreis einer MicroSD-Karte mit 8 GB Speicherplatz beträgt zurzeit ca. 5 €. Obwohl die ehemals tragfähigen Gründe für die TEDS-Strukturen keinen Bestand mehr haben, wurden keine grundsätzlichen Anpassungen vorgenommen. Während bei TEDS explizit zwischen Aktoren und Sensoren unterschieden wird, erfolgt die Bestimmung bei UPnP implizit über die Methodensignaturen. Sensoren können durch den Rückgabewert einer Methode repräsentiert werden. Ebenso ist über die Festsetzung von Methodenparametern Einfluss auf Aktoren zu nehmen. Eine Stärke von TEDS liegt in der Kodierung physikalischer Einheiten. UPnP bietet diese Möglichkeit hingegen nicht. Diesem Umstand wird durch Einbeziehung der Einheitenkodierung in den Sprachumfang von DevXML Rechnung getragen. Da im Zielszenario vernetzte Geräte mit autarker Rechenleistung interagieren, sind formalisierte Kalibrierungsansätze auf Signalebene nicht von Relevanz.

[128] Froitzheim: Multimedia-Kommunikation, S. 113 ff.

Dies liegt darin begründet, dass im Gegensatz zu reinen Messwertaufnehmern bzw. -gebern gemäß IEEE 1451 die interne Weiterverarbeitbarkeit von Signalen vorausgesetzt wird. Zusammenfassend ist festzustellen, dass im Rahmen der Betrachtungsgrenzen TEDS gegenüber UPnP in Kombination mit DevXML keinen Zusatznutzen aufweisen.

Die Technologie Jini erweist sich als unvereinbar mit dem Konzept der generischen Gerätesteuerung. Jini stellt zwar gleichermaßen eine funktionstüchtige Ad-Hoc-Infrastruktur wie UPnP und Zeroconf dar, jedoch führt die Abhängigkeit zur Programmiersprache Java zu enormen Einschränkungen im Hinblick auf potentielle Einsatzbereiche. De facto besteht die Notwendigkeit, die am Jini-Verbund teilnehmenden Geräte mit einer Java Runtime Environment auszustatten, da andernfalls zentrale Zielstellungen wie ortunabhängige Nutzbarkeit und Interaktionsvermögen unterlaufen werden. Mit dieser Anforderung werden offensichtlich Marktgegebenheiten ausgeblendet, denn im Allgemeinen bieten programmierbare Hardwaremodule – beispielsweise das in der Referenzimplementierung vorgestellte Modul der Firma Rabbit Semiconductor – keine Java-Unterstützung. Bei Spezialmodulen mit Java-Unterstützung, die überwiegend im hochpreisigen Segment angesiedelt sind, führt die Java Runtime Environment zu einer Steigerung des Ressourcenbedarfs bei gleichzeitiger Minderung der Performance.

Das zusammenfassende Urteil lautet, dass Jini nicht zu überzeugen vermag und daher von allen weiteren Betrachtungen ausgenommen wird. Vor dem Hintergrund, die Anzahl der zum Einsatz kommenden Protokolle möglichst gering zu halten, wird SNMP und SLP ebenfalls nicht weiter verfolgt.

Geräteidentifikation Die Geräteidentifikation vor Ort findet ebenfalls Eingang in das Konzept. Wahlweise können OCR, RFID und Barcode verwendet werden. Das technologieübergreifende Prinzip besteht in der Zuordnung eindeutiger Gerätekennungen zu den lokal zu erfassenden Identifizierungsobjekten (Schriftzug, RFID-Tag oder Barcode). Die in Kapitel 5 diskutierte Referenzimplementierung umfasst eine RFID-basierte Realisierung.

Audio- und Video-Streaming Die Einbeziehung von AV-Streaminginhalten in das User Interface wird von DevControl unterstützt. Voraussetzung ist deren Bereitstellung über eine konzeptkonforme Schnittestelle. Vor dem Hintergrund, das Spektrum potentieller Endgeräte möglichst breit zu fassen, ist mittels multivalentem Streaming ein MP4-enkodierter Datenstrom über HTTP Live Streaming, ein RA/RV-enkodierter Datenstrom über RTSP/RDP, ein WMA/WMV-enkodierter Datenstrom über MMS oder ein WMA/WMV-enkodierter Datenstrom über HTTP (WMSP) sicherzustellen.

Gerätemodellierung Für die Realisierung der Inter-Gerätekommunikation sowie der Automatisierbarkeit von Abläufen erweist sich die Entkopplung von Gerätefunktionalität und User Interface als zielführend. Dieser Ansatz liegt der Beschreibungssprache DevXML zugrunde, die im Übrigen auch die Autokonfiguration, die Gerätezuordenbarkeit vor Ort und die Geräteüberwachbarkeit vollständig unterstützt.

Die Modellierung von Abhängigkeiten zwischen Gerätefunktionen und -zuständen wird mittels Petrinetzen vorgenommen. Durch die Überführung in relationenalgebraische Notationen können Netzeigenschaften, die für die Gerätesteuerung von essentieller Bedeutung sind, errechnet werden. In diesem Zusammenhang sind die für die Benutzerführung relevante Erreichbarkeitsanalyse und das Berechnungsverfahren zur Identifikation ungültiger Netzzustände zu nennen.

Die formalisierte Beschreibung von funktionalen Abhängigkeiten eignet sich ferner für die geräteübergreifende Modellierung. Funktionen lassen sich neu zueinander in Beziehung setzen und können logisch zu einem „Virtuellen Gerät“ zusammengefasst werden, das sich gegenüber dem Nutzer wie ein reales Gerät darstellt. Mit dem Ziel, grundlegende Funktionalität über eine HTML-Schnittstelle bereitzustellen – und somit das Spektrum potentieller Steuergeräte zu verbreitern – umfasst DevControl parallel zu DevXML die Zugriffmöglichkeit via DevHTML. Die Inter-Gerätekommunikation und die Automatisierbarkeit von Abläufen sind hiermit nicht realisierbar. Abbildung 56 zeigt den zweigliedrigen Ansatz von DevControl unter Bezugnahme auf die zum Einsatz kommenden Basistechnologien. Die Gerätesteuerung auf Basis von DevXML- und DevHTML wird eingehend im Folgekapitel diskutiert.

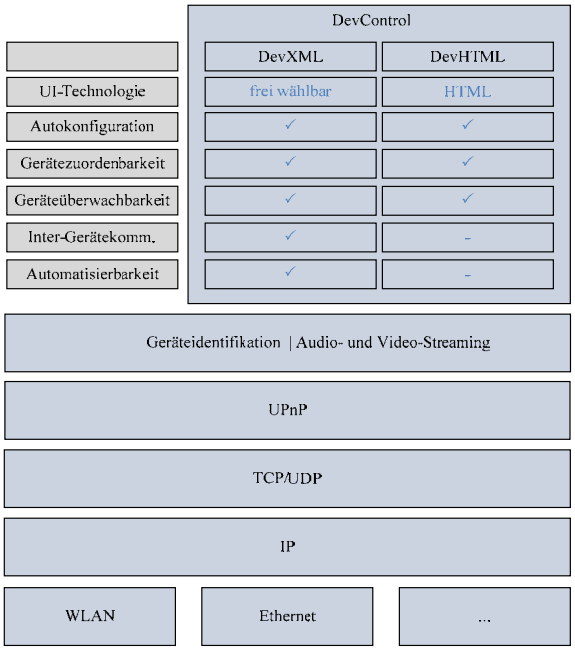


Abbildung 56 Zweigliedriger Ansatz von DevControl. Die Steuerung von Geräten ist mittels DevXML und DevHTML möglich. DevControl setzt auf der Ad-Hoc-Infrastrukturtechnologie UPnP auf. Auf der Transportschicht kommt TCP/UDP und auf der Netzwerkschicht IP zum Einsatz. Für die Bitübertragungs- und Datensicherungsschicht eignen sich beispielsweise WLAN und Ethernet.

4.2 Architektur

Im Folgenden wird die entwickelte Architektur vorgestellt, die alle konzeptrelevanten Technologien mit einbezieht. Sie umfasst die Kommunikationsteilnehmer DevControl-Gerät, DevControl-Gateway und Steuergerät. Geräteidentifikation und AV-Streaming werden über spezielle Ausprägungen von DevControl-Geräten realisiert. Siehe hierzu die Kapitel 4.2.3 „Interface von Mediastreams“ sowie 4.2.4 „Interface von Geräten für die Autoidentifikation“. Mit dem Ziel, möglichst zahlreiche Endgeräte wie PDAs, Mobiltelefone etc. zu grundlegenden Steuerungsaufgaben zu befähigen, wird ein zweigliedriger Ansatz verfolgt. DevControl stützt sich auf die Spezifikationen DevHTML und DevXML.

DevHTML Der Zugriff auf Geräte, bei denen die Voraussetzungen für DevXML nicht gegeben sind – bei Einsatz der Referenzimplementierung z. B. infolge der Nicht-Ausführbarkeit von Silverlight-Code – erfolgt über Web-Interfaces auf Basis von HTML. Diese Benutzerschnittstellen werden über UPnP-Funktionen, die durch das Präfix WI_ zu kennzeichnen sind, bekannt gegeben. Der Rückgabewert stellt einen Link auf eine HTML-Seite dar.

Das DevControl-Gateway verwaltet unter Einbeziehung aller DevXML-Geräte eine Linksammlung und bereitet diese in Form eines HTML-Dokuments, das Steuergeräten über das Protokoll HTTP zugänglich gemacht wird, strukturiert auf. Somit können Steuergeräte unter Nutzung des Protokolls HTTP von DevControl-Geräten die benötigten User Interfaces beziehen und auf deren Grundlage die Steuerung durchführen. Die freie Gestaltbarkeit HTML-basierter User Interfaces führt zu algorithmisch schwer zu fassenden Codeauswertungen und steht in der Folge u. a. einer Inter-Gerätekommunikation entgegen. Die beschriebenen Zusammenhänge werden anhand von Abbildung 57 verdeutlicht.

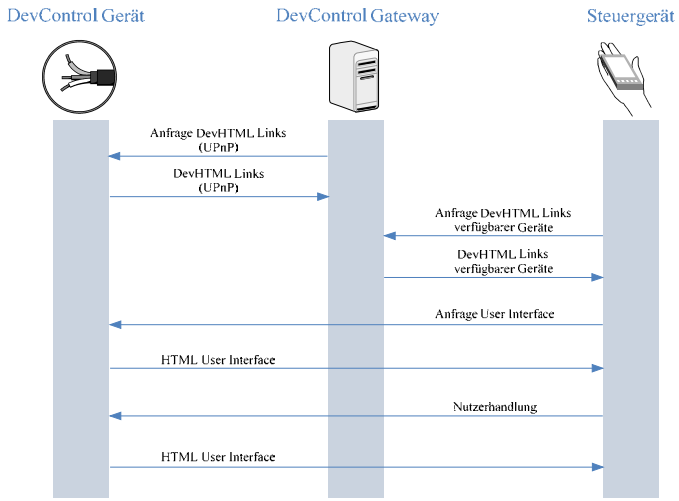


Abbildung 57 DevHTML-basierte Steuerung. Das DevControl-Gateway verwaltet zentral die Web Interface-Links aller im Zugriff befindlichen DevControl-Geräte. Für die Anzeige der jeweiligen Benutzerschnittstelle greifen die Steuergeräte dann auf die vom Gateway aufbereitete Linksammlung zu.

DevXML Bei Einsatz der stark formalisierten Beschreibungssprache DevXML unterliegt das Gesamtsystem hinsichtlich der Inter-Gerätekommunikation und Automatisierung keiner Einschränkung. In diesem Fall wird die Kommunikation mit Präsenzmitteilungen der DevControl-Geräte im Netzwerk unter Verwendung von UPnP eingeleitet. DevControl-Gateways können nun auf die als DevXML-Dokumente vorliegenden formalen Gerätebeschreibungen zugreifen und diese einer Auswertung unterziehen.

DevControl-Gateways zeichnen sich durch die zentrale Verwaltung von eingenommenen Gerätezuständen und Statusvariablen über alle im Netz befindlichen DevControl-Geräte aus. Diese Informationen werden webbasierten Steuergeräten bereitgestellt. Hierzu wird generischer Client-Code – nachfolgend als DevXML-Client bezeichnet – vorgehalten, der den Steuergeräten Zugriff auf die gesamte Gerätefunktionalität ermöglicht. Es eignen sich beispielsweise Ajax (Asynchronous JavaScript and XML) und die in der Referenzimplementierung zum Einsatz kommende Technologie Silverlight. Ein Merkmal von Ajax-Anwendungen ist das sogenannte *XMLHttpRequest*-Objekt, mit dem dynamisch Daten ausgetauscht werden können, ohne einen Seiten-Reload zu bewirken.

In zahlreichen Implementierungen von JavaScript ist das *XMLHttpRequest*-Objekt integraler Bestandteil. Dies trifft nicht auf den Internet Explorer 5.x und 6.x zu. Hier wird die Funktionalität mit dem ActiveX-Objekt vom Typ *Msxml.XMLHTTP* abgebildet. Dies begründet die alternativ gebräuchliche Langform „Asynchronous JavaScript, ActiveX and XML“[129]. Ab IE 7.0 ist das *XMLHttpRequest*-Objekt nativ. In Teilen ist die Interpretation von JavaScript-Code zwar abhängig vom zur Verwendung kommenden Browser, durch übergreifende Bibliotheken wie Prototype und Sarissa können jedoch mögliche Unterschiede auf Entwicklerebene ausgeblendet werden[130].

Letztendlich ist die Wahl auf Silverlight gefallen, um erste Praxiserfahrungen mit der noch jungen, Potential versprechenden Technologie zu gewinnen. Hervorzuheben ist die im .NET-Framework integrierte Komponente LINQ (Language Integrated Query), die das Parsen von XML-Dokumenten mittels *XDocument* auf hoher Abstraktionsebene gestattet. Zudem umfasst Silverlight die low-level-Klasse *XMLReader* zur Bearbeitung von XML-Dokumenten[131]. Für die gängigen Browser IE, Firefox und Safari der Betriebssysteme Windows bzw. Mac OS bietet Microsoft Plugins an. Novell stellt das kompatible Moonlight-Plugin für das Betriebssystem Linux bereit. Die konkrete Ausgestaltung der Nutzerschnittstelle leitet sich aus dem zur Geräteauswahl korrespondierenden DevXML-Dokument ab.

Optional können gestalterische Vorgaben und Elemente, die über den DevXML-Sprachumfang hinausgehen, in das User Interface einbezogen werden. Diese umfassen beispielsweise 3D-Objekte, Schriften mit Farbverläufen und Animationen. UI betreffende Zusatzinformationen müssen von DevControl-Geräten in Form von Silverlight-Code zur Verfügung gestellt werden.

Die DevXML-basierte Steuerung stellt sich wie folgt dar. Zunächst stellt das Gateway sicher, dass es über alle DevXML-Gerätemodelle verfügt. Im Anschluss wird mit dem Steuergerät der DevXML-Client vom Gateway bezogen. Der Nutzer kann dann mit diesem eine Geräteauswahl treffen. Sofern die manuelle Steuerung des Gerätes vorgesehen ist, hat er darüber hinaus die Möglichkeit, feuerbare Transitionen zu selektieren. Das Gateway stellt dem Steuergerät hierbei alle für die Generierung des User Interfaces erforderlichen Informationen zur Verfügung. Sie gehen unmittelbar aus dem betreffenden DevXML-Code hervor. Audio- und Videoechtzeitdaten von DevControl-Mediastreams können direkt eingebunden werden. Die Schnittstellendefinition findet sich in Kapitel 4.2.3.

Optionale User Interfaces für Transitionen werden – sofern vorhanden – bei der Darstellung berücksichtigt. Nutzerhandlungen auf dem Steuergerät führen über das DevControl-Gateway, das wiederum den Aufruf entsprechender Gerätefunktionen initiiert. Die Zustandssynchronisation erfolgt, indem einerseits Änderungen von Gerätezuständen sowie Statusvariablen vom DevControl-Gerät an das Gateway kommuniziert werden und andererseits das Gateway diese Informationen an alle Steuergeräte weiterreicht. Abbildung 58 gibt den Ablauf wieder.

Das zu verwendende Kommunikationsprotokoll zwischen DevXML-Gateway und DevXML-Client wird im Rahmen des Konzeptes nicht vorgegeben, sondern unterliegt im Sinne des technologieoffenen Ansatzes der Implementierungsfreiheit. Hieraus folgt, dass ein DevXML-Client in der Regel nur mit dem DevXML-Gateway interagieren kann, von dem er bezogen wurde.

---

[129] Winkler: JavaScript und Ajax, S. 79 ff.
[130] Crane/Pascarello/James: Ajax in action, S. 132 ff.
[131] Little et al.: Silverlight 3, S. 324 ff.

Eine darüber hinaus gehende Interoperabilität wird nicht gefordert, da bei allen weiteren Betrachtungen angenommen wird, dass in der DevControl-Domäne jeweils nur ein Gateway aktiv ist. Teilfunktionen des DevControl-Gateways können – alternativ zu obigen Darstellungen – auch direkt vom DevXML-Client übernommen werden. Dies setzt eine Verlagerung von Programmlogik voraus. Ein DevXML-Client, der über einen UPnP-Protokollstapel verfügt, kann auf diese Weise z. B. unter Umgehung des Gateways direkt Funktionsaufrufe auf einem DevControl-Gerät initiieren. Aus den Architekturvorgaben resultierende Anforderungen an DevControl-Geräte und DevControl-Gateways sind in den Kapiteln 4.2.1 und 4.2.2 beschrieben.

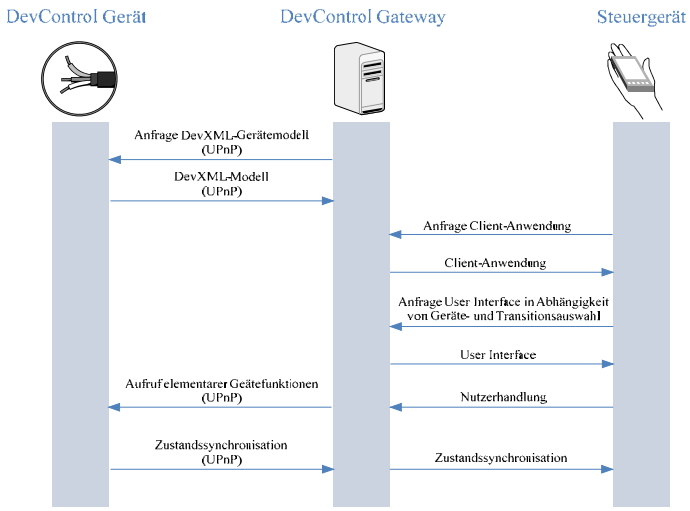


Abbildung 58 DevXML-basierte Steuerung. Das DevControl-Gateway greift mittels UPnP auf DevXML-Code der DevControl-Geräte zu und bereitet die Daten so auf, dass Steuergeräte ohne zusätzliche Software die zugehörigen User Interfaces darstellen können.

Der Einsatz einer Terminalanwendung zum Zwecke der Gerätesteuerung ist naheliegend. Die Grundidee besteht in der Übertragung von Bildschirminhalten eines lokalen Steuergerätes auf einen entfernten Client, der wiederum entgegengenommene Steuerbefehle zurückübermittelt. Geräte, die auf Grundlage dieses Ansatzes eingebunden werden, weisen gegenüber DevControl-Geräten Nachteile auf. Hierzu zählen z. B. Ressourcenintensität durch kontinuierlichen Datenverkehr und mangelnde Inter-Geräte-Kommunikation, d. h. die Unvereinbarkeit mit dem Konzept der virtuellen Geräte. Damit kann eine Terminalanwendung im Allgemeinen DevControl nicht ersetzen. Sie bietet jedoch die Möglichkeit, nicht DevControl-konforme Geräte einzubinden und deren Grundfunktionalität zu nutzen. Dies zeigt die in Kapitel 6.5 vorgestellte Entwicklung „Terminal“.

4.2.1 Interface von DevControl-Geräten

Die infrastrukturellen Konzeptvorgaben lassen auf der Netzzugangsebene beispielsweise ein Ethernet-Anschluss oder ein WLAN-Modul zu. Alle Funktionen und Statusvariablen, die Teil der Gerätemodellierung auf Basis von DevXML sind, müssen über UPnP zugänglich gemacht werden. Statusvariablen sind hierbei durchgängig mit dem Attribut „evented“ zu versehen. Wie in Kapitel 2.2.3 erläutert, werden Funktionen (= Aktionen) auf Service-Ebene zusammengefasst. Die von einem DevControl-Geräte zu implementierenden Dienste und Aktionen stellt nachfolgendes Klassendiagramm dar.

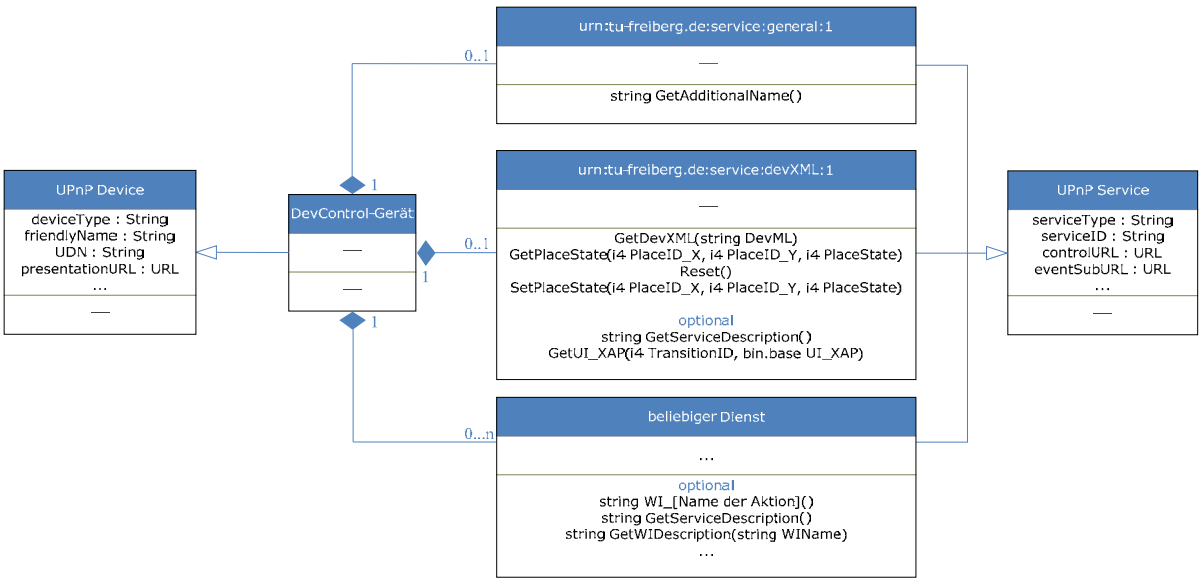


Abbildung 59 Klassendiagramm eines DevControl-Gerätes. Der Dienst *urn:tu-freiberg.de:service:general:1* stellt optional einen Gerätezusatznamen bereit. Im Dienst *urn:tu-freiberg.de:service:devXML:1* ist die mit DevXML verbundene Funktionalität zusammenzufassen. Weitere Aktionen inklusive DevHTML-Webinterfaces können in beliebigen Diensten implementiert werden.

Soll das DevControl-Gerät über einen Zusatznamen verfügen, ist die Aktion *string GetAdditionalName()* innerhalb des Dienstes *urn:tu-freiberg.de:service:general:1* zu implementieren. Hiermit wird die Zuordenbarkeit von Geräten erleichtert. Zusatznamen sind eindeutig zu wählen. Im Rahmen der Definition von Virtual Devices können sie darüber hinaus für die Gerätereferenzierung herangezogen werden.

Gegenüber der alternativen Verwendung von *Unique Device Names* haben sie den Vorteil, dass sie frei gewählt werden können und eine Anpassung nur im Bedarfsfall erfolgt. *Unique Device Names* hingegen werden vom UPnP-Framework der Firma Intel bei jedem Gerätestart neu generiert. Bei deren Verwendung ist die zum Virtual Device gehörige DevXML-Datei somit fortlaufend zu aktualisieren.

Aktionen, die mit der Grundfunktionalität DevXML-basierter Steuerung unmittelbar in Zusammenhang stehen, sind dem Dienst *urn:tu-freiberg.de:service:devXML:1* zuzuordnen. *GetDevXML(string DevXML)* stellt die DevXML-Gerätebeschreibung bereit. Die Aktion *GetPlaceState(i4 PlaceID_X, i4 PlaceID_Y, i4 PlaceState)* liefert den Status des Platzes mit der ID (*ID_X, ID_Y*) zurück. Ist der Status nicht definiert, so ist -1 zurückzugeben. Siehe hierzu auch Kapitel 4.3.2.1. Nach Aufruf von *Reset()* sind sämtliche Statusinformationen über Plätze zu löschen und das DevControl-Gerät in den Ursprungszustand zu überführen. Ferner wird das DevControl-Gateway hiermit angewiesen, das initiale DevXML-Netz darzustellen. Die Aktion *SetPlaceState(i4 PlaceID_X, i4 PlaceID_Y, i4 PlaceState)* setzt den Status des Platzes mit der ID (*ID_X, ID_Y*). Der Aufruf erfolgt durch das DevControl-Gateway, das die aktuellen Zustände verwaltet. Mit dem Netzabbild auf dem DevControl-Gerät ist sichergestellt, dass auch nach einem Neustart des Gateways – z. B. infolge von Netzwerkstörungen – weiterhin Informationen über den aktuellen Gerätezustand vorliegen. Wird innerhalb der DevXML-Gerätebeschreibung auf Silverlight-basierte optionale UI-Vorlagen verwiesen, ist innerhalb des Services *urn:tu-freiberg.de:service:devXML:1* darüber hinaus die Aktion *GetUI_XAP(i4 TransitionID, bin.base64 UI_XAP)* zu implementieren.

Sie liefert den Silverlight-Applikationscode des UI für die Transition mit der ID *TransitionID* zurück. Die Projektdatei muss im kompilierten XAP-Format[132] vorliegen. Eine UI-Vorlage, für dessen Erstellung sich auch Designwerkzeuge Dritter eignen, bezieht sich auf einen Zustandsübergang. Dieser entspricht in der Petrinetztheorie einer Transition. Nähere Ausführungen hierzu folgen in Kapitel 4.3.

Für die HTTP-basierte Steuerung sind UPnP-Aktionen mit dem Präfix WI_ und der Signatur *string WI_[Name der Aktion]()* zu definieren. Als Rückgabewert der Aktion ist ein Link auf eine HTML-Seite zu liefern, auf deren Grundlage die Interaktion zwischen steuerndem und zu steuerndem Gerät erfolgt. Die Definition von WI_-Aktionen kann innerhalb frei wählbarer Dienste vorgenommen werden. Optional sind über die Aktion *string GetServiceDescription*() Dienstebezeichnungen zuordenbar. Gleiches gilt für ergänzende Bezeichnungen von Webinterfaces. Spezifikationsgemäß ist das Ergebnis von *string GetWIDescription(string WIName)* parameterabhängig. *WIName* nimmt hierbei den Namen eines Webinterfaces, d. h. *WI_[Name der Aktion]*, auf.

4.2.2 Interface von DevControl-Gateways

Webbasierten Steuergeräten sind durch das DevControl-Gateway Zugangsmöglichkeiten via DevHTML und DevXML zu bieten. Der DevHTML-Link ist frei wählbar und von der UPnP-Aktion *string GetDevHTMLUrl()* innerhalb des Service *urn:tu-freiberg.de:service:display:1* als Rückgabewert zu liefern. Dies ermöglicht dessen automatische Erfassung mittels Browser-Plugin. In der Referenzimplementierung lautet der entsprechende Link *http://[hostname]:8181/cp/ShowDevices.html*.

Analog ist ein Einstiegspunkt für die Nutzung von Gerätefunktionalität auf Basis von DevXML vorzusehen. Der zugehörige Link ist wiederum frei wählbar und von der UPnP-Aktion *string GetDevXMLUrl()* innerhalb des Service *urn:tu-freiberg.de:service:display:1* zurückzuliefern. Hiermit ist auch dessen automatische Erfassung mittels Browser-Plugin möglich. In der Referenzimplementierung erfolgt der Zugang über die Adresse *http://[hostname]:8181/parser/Navigator.html*. Abbildung 60 zeigt das Klassendiagramm eines DevControl-Gateways.

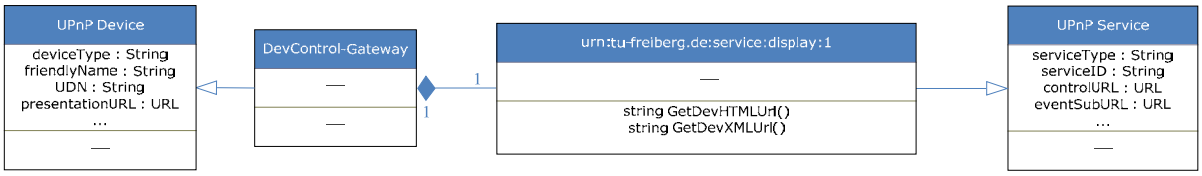


Abbildung 60 Klassendiagramm eines DevControl-Gateways. Der Link für den DevHTML-Zugangspunkt wird von der Aktion *string GetDevHTMLUrl()* zurückgeliefert. Der DevXML-Zugangspunkt wird analog über die Aktion *string GetDevXMLUrl()* bereitgestellt.

[132] Anderson: Pro Business Applications with Silverlight 4, S. 32.

Zwischen dem DevControl-Gateway und dem von ihm bereitgestellten DevXML-Client muss eine Synchronisation erfolgen. Nur so ist gewährleistet, dass die Zustandsänderung eines DevControl-Gerätes, z. B. infolge einer Schalthandlung, bei allen Steuergeräten im Netzverbund zu einer entsprechenden Anpassung der Benutzerschnittstelle führt. Durch das DevControl-Gateway sind unter Rückgriff auf den Mechanismus des UPnP-Event-Listening Änderungen aller Statusvariablen der im Netzverbund befindlichen DevControl-Geräte zu verfolgen. Da der DevXML-Client als funktionaler Bestandteil des DevControl-Gateways aufgefasst werden kann, ist die Realisierung des Event-Listening alternativ über diesen möglich. Vergleiche hierzu die abschließenden Ausführungen in Kapitel 4.2.

4.2.3 Interface von Mediastreams

Die Möglichkeit der Einbindung eines Mediastreams mittels DevXML in das User-Interface eines beliebigen DevControl-Gerätes basiert auf der Vergabe einer eindeutigen Bezeichnung. Hierzu ist innerhalb des Dienstes *urn:tu-freiberg.de:service:general:1* die Aktion *string GetAdditionalName()* bereitzustellen. Unter Rückgriff auf diese Methode kann eine über das DevXML-Element *<camera>* referenzierte Webcam im Netzwerk identifiziert werden. Vergleiche hierzu die Ausführungen in Kapitel 4.3.2.2. Im Gegensatz zu sonstigen DevControl-Geräten ist die Aktion *string GetAdditionalName()* bei Mediastreams obligatorisch (vgl. Kapitel 4.2.1).

Die Bereitstellung eines Medienstroms erfordert die Implementierung des UPnP-Dienstes *urn:tu-freiberg.de:service:mediastream:1* mit den WI_-Aktionen *string WI_GetJpgImage()*, *string WI_GetMP4oHLS()*, *string WI_GetRVoRTSP()*, *string WI_GetWMVoMMS()* und *string WI_GetWMVoHTTP()*. Die Rückgabewerte beschreiben Links für den Medienabruf. In der Reihenfolge der obigen Nennung lauten die zugrunde liegenden Datenformate und Übertragungsprotokolle JPG über HTML, MP4 über HTTP Live Streaming, RV über RTSP, WMV über MMS sowie WMV über HTTP.

Mindestens eine der genannten Kombinationen muss der mit dem Dienst verbundene Medienserver unterstützen. Die Nichtunterstützung eines Datenformat-Übertragungsprotokoll-Paares ist durch eine leere Zeichenkette zu kennzeichnen. Es liegt außerhalb der Spezifikation, ob der Dienst mit einem integrierten oder separaten Medienserver zusammenwirkt. Das vorliegende Konzept der abstrahierten Funktionsdarbietung lässt beide Varianten zu. Abbildung 61 zeigt das Klassendiagramm eines Mediastreams.

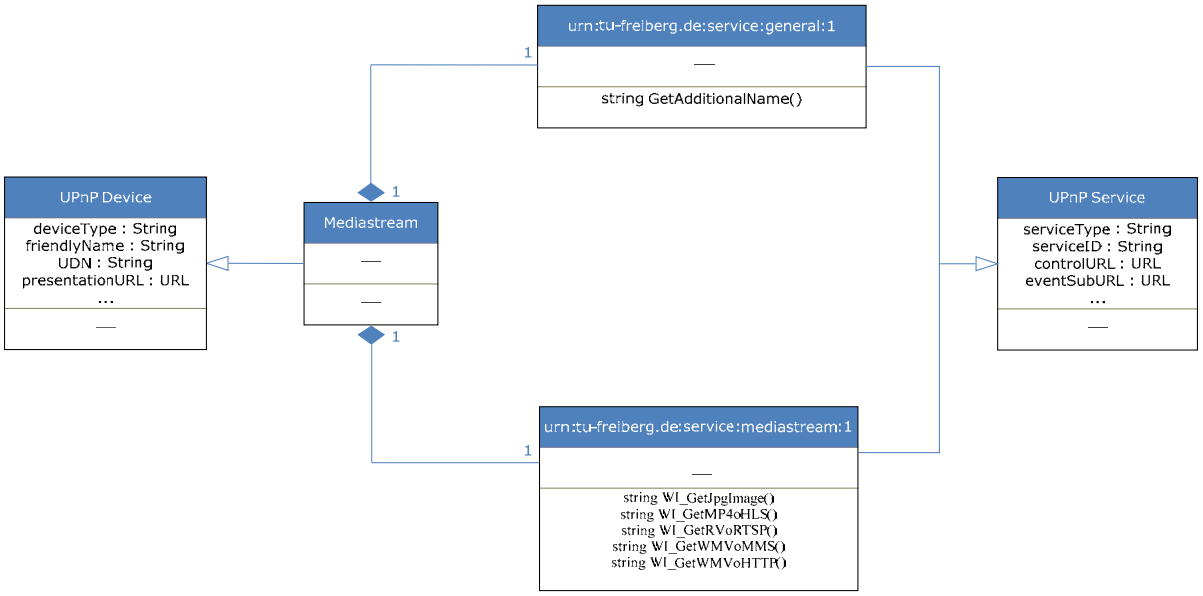


Abbildung 61 Klassendiagramm eines Mediastreams, dessen eindeutige Bezeichnung von der Aktion *string GetAdditionalName()* innerhalb des Dienstes *urn:tu-freiberg.de:service:general:1* zurückgegeben wird. Der Dienst *urn:tu-freiberg.de:service:mediastreams:1* umfasst mehrere Aktionen, die den Zugriff auf den Medienstream mit unterschiedlichen Protokollen gestatten.

4.2.4 Interface von Geräten für die Autoidentifikation

In Kapitel 2.3 wurden OCR, RFID und Barcode vorgestellt. Diese gängigen Verfahren eignen sich für die Autoidentifikation von Geräten. Als Schnittstelle für den verfahrensunabhängigen Zugriff ist einheitlich der Dienst *urn:tu-freiberg.de:service:autoid:1* mit der Aktion *string GetData()* zu implementieren. DevControl-Gateways greifen in regelmäßigen Abständen hierauf zu. Befindet sich ein DevControl-Gerät mit entsprechender Kennzeichnung in Reichweite des Lesegerätes, so ist der *Unique Device Name* zurückzuliefern. Abbildung 62 stellt das zugehörige Klassendiagramm dar.

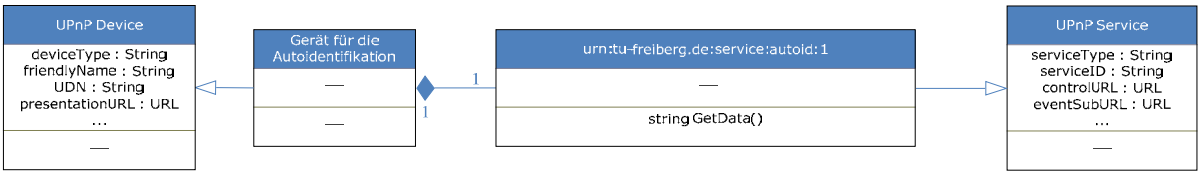


Abbildung 62 Klassendiagramm eines Gerätes für die Autoidentifikation. Befindet sich in dessen Reichweite eine entsprechende Kennzeichnung, so wird der darauf befindliche *Unique Device Name* von der Aktion *string GetData()* zurückgegeben.

Auf dieser Grundlage wird im User Interface des aktuell verwendeten DevHTML- bzw. DevXML-Control Points die Geräteauswahl getroffen. Lesegeräte können alternativ zur Geräteerkennung für andere Zwecke eingesetzt werden. In diesem Fall ist applikationsabhängig zu vereinbaren, wie die von *string GetData()* zurückgegebene Zeichenkette zu interpretieren ist. Bei dem in Kapitel 6.8.1 beschriebenen Virtual Device „Haptic Phone“ ist auf dem RFID-Tag beispielsweise kein *Unique Device Name*, sondern eine Rufnummer gespeichert.

4.3 DevXML

In den folgenden Unterkapiteln wird detailliert auf den XML-Dialekt DevXML eingegangen. Am Anfang steht die ausführliche Beschreibung des Datenmodells. Die Erläuterung der DevXML-Sprachelemente, die das Fundament für die Gerätemodellierung bilden, schließt sich in Kapitel 4.3.2 an. Der formalisierte Modellierungsansatz mit mathematischem Zugang stellt im Weiteren den Ausgangspunkt für die Navigation innerhalb einer Zustandsmenge dar. Das zugrunde liegende Verfahren, bei dem der Gerätenutzer die Zieltransition vorgibt, wird in Unterkapitel 4.3.3 vorgestellt. Des Weiteren wird in Unterkapitel 4.3.4 dargelegt, wie das Einnehmen ungültiger Gerätezustände wirksam und effizient unterbunden werden kann.

4.3.1 Datenmodell

Die Zuweisung von Werten zu DevXML-Attributen bzw. Elementen erfolgt wahlweise direkt oder über Variablen. In Abhängigkeit des jeweiligen DevXML-Sprachelements ist der Datentyp entweder explizit zu spezifizieren (siehe Element <*param*>) oder er geht implizit aus dem Kontext hervor (siehe z. B. Element <*preedge*>). Folgende Parametertypen sind innerhalb von DevXML gültig:

Tabelle 20 Parametertypen innerhalb von DevXML. Es werden Ganzzahlen, Fließkommazahlen, Zeichenketten sowie Wahrheitswerte unterstützt.

Parametertyp	Beschreibung
ui1	unsigned 1-Byte Integer
ui2	unsigned 2-Byte Integer
ui4	unsigned 4-Byte Integer
i1	signed 1-Byte Integer
i2	signed 2-Byte Integer
i4/int	signed 4-Byte Integer
r4/float	4-Byte Fließkommazahl
r8	8-Byte Fließkommazahl
char	Unicode Zeichenkette der Länge 1
string	Unicode Zeichenkette beliebiger Länge
boolean	Wahrheitswert

Für die Darstellung von Tabellendaten mit einem Steuerelement ist die DevXML-Datenstruktur *table* einzusetzen. Siehe hierzu auch die Ausführungen zum Element <*controlelement*>. Die *table*-Datenstruktur besitzt nachstehenden Aufbau.

Quellcode 1 Table-Datenstruktur

```
<result>
  <vector>
    <element name=c1> r11 </element>
    <element name=c2> r12 </element>
    <element name=cm> r1m </element>
  </vector>

...

 <vector>
    <element name=c1> rn1 </element>
    <element name=c2> rn2 </element>
    <element name=cm> rnm </element>
  </vector>
</result>
```

Die Matrix wird über n Zeilenvektoren *<vector>* beschrieben. Komponentenweise ist der mit einem Eintrag verbundene Spaltenname über das Attribut *name* des Elements *<element>* anzugeben. Die Vektoren mit jeweils m Elementen führen zu den Datenfeldern r_{ij} (mit $i = 1...n$, $j = 1...m$). Somit ergibt sich die in Tabelle 21 abgebildete Matrixdarstellung der Datenstruktur bestehend aus n Zeilen und m Spalten.

c_1	c_2	...	c_m
r_{11}	r_{12}	...	r_{1m}
r_{21}	r_{22}	...	r_{2m}
...	...	...	...
r_{n1}	r_{n2}	...	r_{nm}

Tabelle 21 Matrixdarstellung der Datenstruktur des Typs Table, die aus n Zeilenvektoren mit je m Einträgen besteht.

4.3.2 Sprachelemente

Am besten ist die Struktur der Sprachelemente von DevXML anhand von Abbildung 64 zu erkennen. Die Elemente werden nachfolgend den drei Kategorien

- Grundlegende Petrinetzstrukturelemente,
- Elemente zur Platzspezifikation und
- Elemente zur Transitionsspezifikation

zugeordnet. In den gleichlautenden Unterkapiteln werden syntaktische und semantische Spracheigenschaften vermittelt. Für jedes Element sind Informationen zu Kardinalität, Kindelementen und Attributen explizit aufgeführt. Die Zusammenhänge werden durchgängig anhand von Diagrammen veranschaulicht. Diese wurden mit dem Programm „Altova xmlspy 2009“ erstellt. Für die grafische Darstellung eines XML-Elementes e_1 wird ein Rechteck verwendet, dessen Inneres die Bezeichnung des Elementnamens beinhaltet. Die dem Element zugehörigen Attribute $a_1, a_2, ..., a_m$ sind in einer Registerform aufgeführt.

Die Zuordnung wird über eine Verbindungslinie hergestellt. Verfügt das Element e_1 über Subelemente $s_1, s_2, ..., s_n$, wird dies durch eine Verbindungslinie kenntlich gemacht, die von e_1 zu einem Achteck führt und dort weiter in Richtung Subelemente $s_1, s_2, ..., s_n$ verzweigt. Ist ein XML-Element oder -Attribut erforderlich, so ist die umgebende Rahmenlinie durchgezogen.

Ein optionales Element oder Attribut ist hingegen durch eine gestrichelte Rahmenlinie gekennzeichnet. Rechteck-, Register-, und Achteckform besitzen rechtsseitig ein „+"-Zeichen, wenn dazugehörige Teilinformationen ausgeblendet sind. Ein „-"-Zeichen signalisiert die vollumfängliche Informationsdarstellung. Die Symbolik ist in Abbildung 63 zu erkennen. Mit Ausnahme von a_1 und s_2 sind im Beispiel alle Attribute und Elemente obligatorisch.

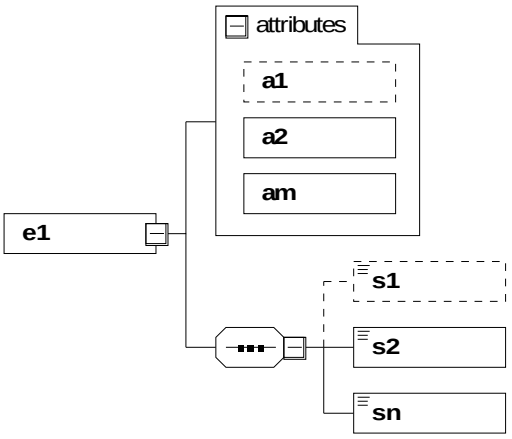


Abbildung 63 Symbolik zur Visualisierung von XML-Sprachelementen. Das Element e_1 verfügt über die Attribute $a_1, a_2, ..., a_m$ und die Subelemente $s_1, s_2, ..., s_n$. Die Rahmenlinie eines erforderlichen Attributs oder Elements ist durchgezogen.

Abbildung 64 gibt eine Übersicht der Sprachelemente. Das im Anhang A1 beigefügte XML-Schema beinhaltet alle syntaxrelevanten Vorgaben. Es gestattet die automatisierte Prüfung von DevXML-Dokumenten auf Gültigkeit.

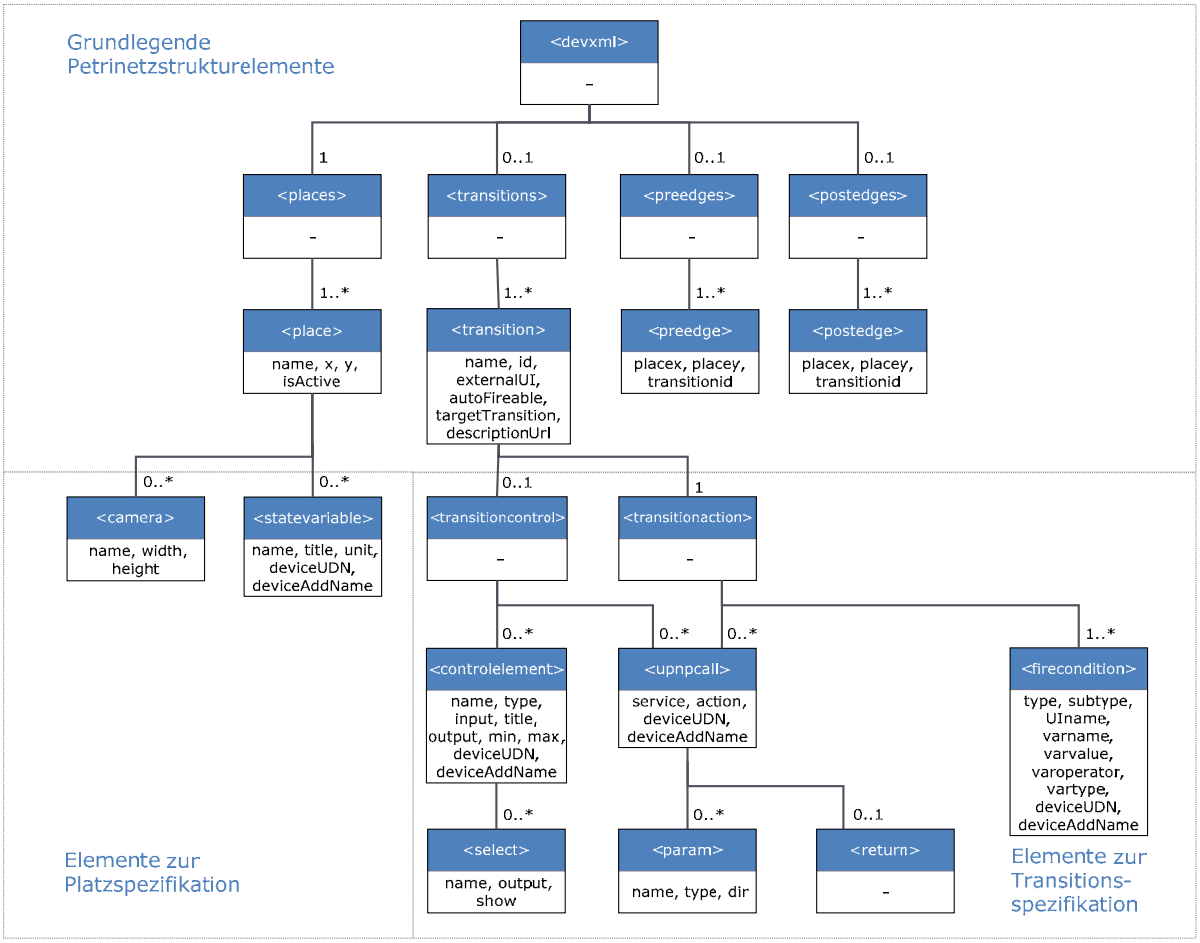


Abbildung 64 Sprachelemente von DevXML. Sie werden den Kategorien „Grundlegende Petrinetzstrukturelemente“, „Elemente zur Platzspezifikation“ und „Elemente zur Transitionsspezifikation“ zugeordnet. An dieser Untergliederung orientieren sich die weiteren Ausführungen.

4.3.2.1 Grundlegende Petrinetzstrukturelemente

Die Grundkonzepte der Petrinetztheorie sind in die Entwicklung der Beschreibungssprache DevXML eingeflossen. Gerätezustände werden über Plätze und Gerätefunktionen über Transitionen abgebildet. Ein Gerätezustand ist hierbei über die Markierung des Netzes definiert. Ein Geräteteilzustand entspricht einer einzelnen Platzmarkierung, die alternativ den Wahrheitswert *true* oder *false* annehmen kann. Pre- und Postkanten bestimmen den Markenfluss, das heißt, sie beschreiben die durch Funktionsausübung bedingte Überführung in den Folgezustand. Für die Modellierung eignen sich binäre Petrinetze, bei der die Kantenwichtung keinen Einfluss auf das Netzverhalten hat. Definition 1 vereinfacht sich somit zu:

$$N = (P, T, Pre, Post, M_0) \tag{4.1}$$

Hiervon leitet sich die DevXML-Struktur in den beiden Ebenen unterhalb des Wurzelelements <*devxml*> ab. Die Elemente <*places*>, <*transitions*>, <*preedges*> und <*postedges*> korrespondieren unmittelbar mit den ersten vier Einträgen des Tupels *N*. Die Anfangsmarkierung M_0 wird über das Attribut *isActive* der <*place*>-Elemente kodiert. Ein DevXML-Dokument ist durch einmaliges Vorkommen des root-Elements <*devxml*>, das die oben bereits erwähnten Elemente <*places*>, <*transitions*>, <*preedges*> und <*postedges*> unmittelbar umschließt, gekennzeichnet. Siehe hierzu Tabelle 22.

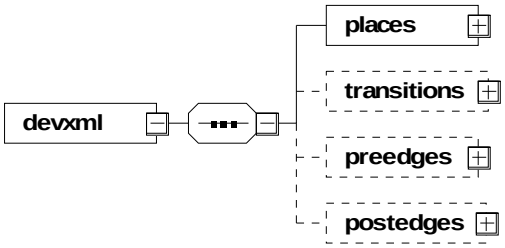
Diagramm	
Kardinalität	1
Kindelemente	places, transitions, preedges, postedges
Attribute	-

Tabelle 22 Darstellung des DevXML-Elements <*devxml*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Nachfolgend wird dargelegt, warum die Petrinetzkomponenten Platz, Transition, Prekante und Postkante innerhalb von DevXML auf einer Hierarchieebene modelliert werden und die Bildung von Strukturen, die an die Objektorientierung anlehnen, unterbleibt. Vielmehr werden Petrinetzkomponenten unter Nutzung von eindeutigen Bezeichnern in Beziehung zueinander gesetzt. Die Abbildung von Plätzen als Subelemente von Pre- bzw. Postkanten würde aufgrund der hierzu bestehenden 1:m-Beziehungen zu Redundanzen führen. Infolgedessen werden Plätze als Hauptelemente ausgeprägt. Vor dem Hintergrund der Redundanzvermeidung erweist sich die Beschreibung von Transitionen als Subelement von unmittelbar hiermit verknüpften Pre- oder Postkanten als ungeeignet. Auch hier liegen 1:m-Beziehungen vor. Prekanten könnten redundanzfrei als Unterstruktur eines Platzes oder einer Struktur ausgeführt werden. Jeder Prekante ist genau ein Platz als Start- und eine Transition als Endpunkt zugeordnet.

Die Wahl zwischen Platz und Transition als umschließenden Element wäre jedoch willkürlich, da diese relationenalgebraisch betrachtet gleichwertig sind. Denn die Menge der Plätze P und die Menge der Transitionen T zählen gleichsam zur Knotenmenge K (vgl. Definition 2). Im Sinne der Konsistenzwahrung werden Prekanten daher ebenfalls separat modelliert. Die Argumentation trifft für Postkanten in analoger Weise zu. Einer ähnlichen Strukturierung folgt das an der Humboldt Universität zu Berlin entwickelte Petrinetz-Austauschformat *Petri Net Markup Language* (*PNML*)[133]. Das Element <*places*> mit der Kardinalität 1 ist obligatorisch, da Geräte stets zustandsbehaftet sind. Dem Element <*places*> ist das Kindelement <*place*> zugeordnet. Tabelle 23 gibt die Zusammenhänge in der Übersicht wieder.

[133] Iso/Iec: Software and Systems Engineering – High-level Petri Nets, Part 2: Transfer Format, ISO/IEC 15909-2.

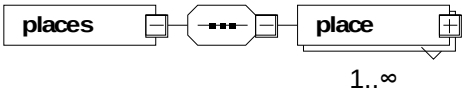
Diagramm	
Kardinalität	1
Kindelemente	place
Attribute	-

Tabelle 23 Darstellung des DevXML-Elements <*places*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Das Element <*place*> kommt beliebig häufig, mindestens jedoch einmal vor. Der Wert des beinhalteten *name*-Attributs wird bei der Visualisierung der Gerätemodellierung als Platzbezeichnung angegeben. Die Attribute *x* und *y* stellen ein Tupel dar, das als eindeutiger Platzidentifizierer dient. Für die Komponenten ist der Wertebereich von 0 bis 32.767 zugelassen. Gleichzeitig geben die Attribute die jeweilige Platzpositionierung in der Ebene wieder. So kann Einfluss auf die darstellerische Ausgestaltung der Geräteabhängigkeiten genommen werden. Ist dem Attribut *isActive* der Wahrheitswert *true* zugewiesen, gilt der zugehörige Platz als markiert. Der Wahrheitswert *false* indiziert einen demarkierten Platz. Die Diskussion der Kindelemente <*camera*> und <*statevariable*> ist Gegenstand des Kapitels Elemente zur Platzspezifikation. Beachte Tabelle 24.

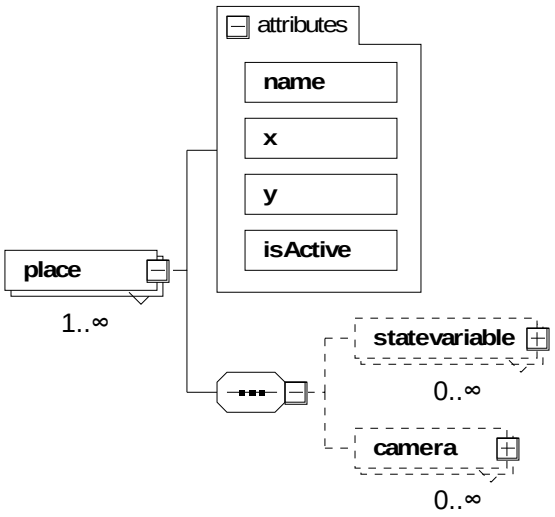
Diagramm	
Kardinalität	1...∞
Kindelemente	statevariable, camera
Attribute	erforderlich: name, x, y, isActive

Tabelle 24 Darstellung des DevXML-Elements <*place*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Möglicherweise ist ein Platz für die Beschreibung eines Gerätes ausreichend. In diesem Fall bedarf die Modellierung keiner Angabe von Transitionen, Pre- und Postkanten. Der fakultative Charakter kommt durch die Kardinalität 0...1 bei den Elementen <*transitions*>, <*preedges*> und <*postedges*> zum Ausdruck. Die Menge der Transitionen wird durch das Element <*transitions*>, das wiederum das Kindelement <*transition*> beinhaltet, begrenzt (siehe Tabelle 25).

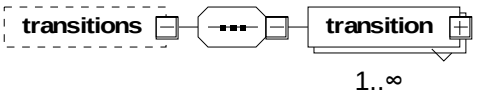
Diagramm	
Kardinalität	0...1
Kindelemente	transition
Attribute	-

Tabelle 25 Darstellung des DevXML-Elements *<transitions>* mit Angaben zu Kardinalität, Kindelementen und Attributen.

Über das Attribut *name* des Elements *<transition>* ist einer Transition eine Bezeichnung zuzuordnen. Hierüber wird im späteren Steuerungsprozess die dargebotene Gerätefunktionalität qualifiziert. Die Vergabe einer eindeutigen Identifikationsnummer über das Attribut *id* ist notwendig, damit eine Transition als Ankerpunkt von Pre- und Postkanten referenziert werden kann. Der Wert ist aus dem Bereich 0 bis 32.767 zu wählen.

Liegt eine externe Beschreibung des transitionsspezifischen UI in Form von Silverlight-Code vor, ist dem optionalen Attribut *externalUI* der Wahrheitswert *true* zuzuweisen. Hiermit wird ein DevControl-Gateway angewiesen, unter Verwendung der Methode *GetUI_XAP(i4 TransitionID, bin.base64 UI_XAP)* eine kompilierte DLL-Klassendatei zu beziehen (vergleiche Kapitel 4.2.1). Als Methodenparameter ist von diesem die oben beschriebene Identifikationsnummer *id* zu übergeben. Es muss beachtet werden, dass die Einbindung einer externen UI-Beschreibung an Vorgaben hinsichtlich Bereitstellung und Benennung von UI-Elementen geknüpft ist. Im Rahmen der Ausführungen zu den Kindelementen *<transitioncontrol>* und *<transitionaction>* in Kapitel 4.3.2.3 wird hierauf vertiefend eingegangen.

Mit dem Attribut *autoFireable* wird spezifiziert, ob ein DevXML-Control Point im Rahmen der geführten Navigation die Feuerung der betreffenden Transition ohne Nutzereingriff bewirken kann. Dies ist bei Zuweisung des Wahrheitswertes *true* der Fall[134]. Der standardmäßig zugewiesene Wert *false* hingegen unterbindet diesen Automatismus.

Mit dem Attribut *targetTransition* wird bestimmt, ob eine Transition innerhalb eines DevXML-Control Points bei der geführten Navigation für den Nutzer auswählbar ist. Per Konvention gilt der Wahrheitswert *true* als Voreinstellung. Die Unterdrückung einer Transition im Auswahlmenü kann durch explizite Zuweisung des Wahrheitswertes *false* erwirkt werden. Letztgenannte Möglichkeit ist beispielsweise dann sinnvoll, wenn eine Transition nur im Zusammenhang mit der Feuerung einer Sequenz eine Rolle spielt[134]. Mit dem optionalen Attribut *descriptionUrl* kann ein Link auf eine Webseite gesetzt werden, die beispielsweise Informationen zur Wirkweise der betreffenden Transition beinhaltet. Tabelle 26 stellt das Element *<transition>* im Überblick dar.

[134] vgl. hierzu den DevXML-Code des Gerätes Jukebox.

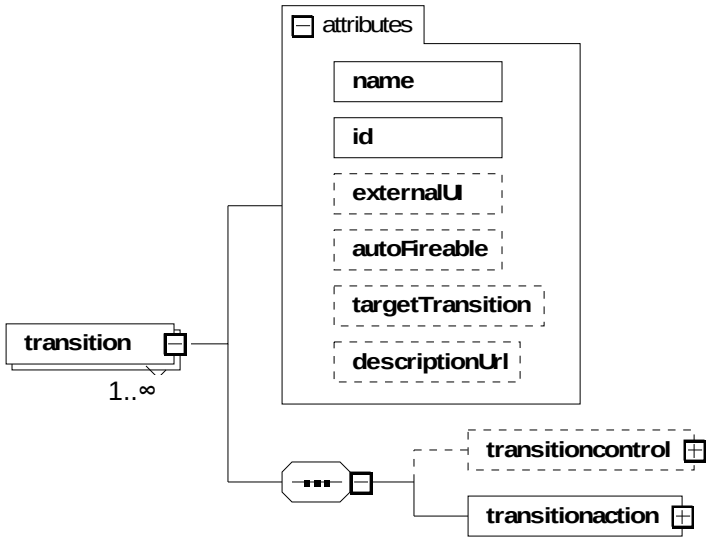
Diagramm		
Kardinalität	1...∞	
Kindelemente	transitioncontrol, transitionaction	
Attribute	erforderlich: optional:	name, id externalUI, autoFireable, targetTransition, descriptionUrl

Tabelle 26 Darstellung des DevXML-Elements <*transition*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Die Menge der Prekanten wird mit dem Element <*preedges*> ausgezeichnet. Für die Beschreibung einer einzelnen Prekante existiert das Kindelement <*preedge*> (siehe Tabelle 27).

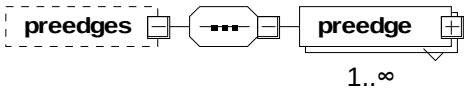
Diagramm		
Kardinalität	0...1	
Kindelemente	preedge	
Attribute	-	

Tabelle 27 Darstellung des DevXML-Elements <*preedges*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Ein Platz wird als Startpunkt einer Prekante festgelegt, indem der Platzidentifizierer *(x,y)* komponentenweise den Attributen *placex* und *placey* des Elements <*preedge*> zugewiesen wird. Die Auswahl einer Transition als Endpunkt erfolgt über das Attribut *transitionid* unter Angabe der zugehörigen Identifikationsnummer. Die Zusammenhänge sind in nachfolgender Tabelle zusammengefasst.

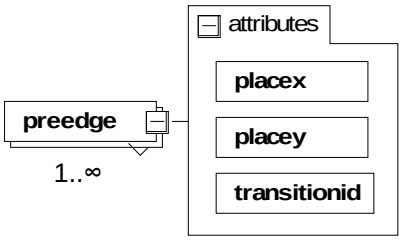
Diagramm		
Kardinalität	1...∞	
Kindelemente	-	
Attribute	erforderlich:	placex, placey, transitionid

Tabelle 28 Darstellung des DevXML-Elements <*preedge*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Postkanten werden in Analogie zu Prekanten modelliert. Hierzu stehen das aggregierende Element <*postedges*> und dessen Kindelement <*postedge*> zur Verfügung (siehe Tabelle 29).

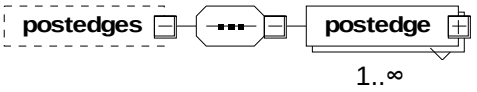
Diagramm	
Kardinalität	0...1
Kindelemente	postedge
Attribute	-

Tabelle 29 Darstellung des DevXML-Elements <*postedges*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Das Element <*postedge*> besitzt den gleichen Aufbau wie das Element <*preedge*>. Die Attribute *placex*, *placey* und *transitionid* referenzieren ebenfalls einen Platz sowie eine Transition. Im Gegensatz zu einer Prekante stellt bei einer Postkante die Transition den Start- und der Platz den Endpunkt dar. Tabelle 30 stellt das Beschriebene zusammenfassend dar.

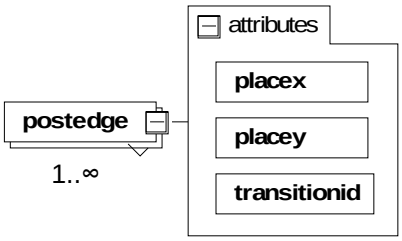
Diagramm		
Kardinalität	1...∞	
Kindelemente	-	
Attribute	erforderlich:	placex, placey, transitionid

Tabelle 30 Darstellung des DevXML-Elements <*postedge*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

4.3.2.2 Elemente zur Platzspezifikation

In dem vorangegangenen Kapitel wurde gezeigt, wie mit DevXML-Standardelementen die Modellierung von grundlegenden Geräteabhängigkeiten vorgenommen wird. Die nähere Spezifikation von Geräten, die über den reinen Aufbau von Petrinetzstrukturen hinausgeht, ist mit den Kindelementen <*camera*> und <*statevariable*> möglich. Zum Spezifikationsumfang zählen die Einbindung von Streaminginhalten und die Anzeige von mit dem Gerätezustand verbundenen Variablen. Die Kardinalität des für audiovisuelle Streaminginhalte zu verwendenden Elements <*camera*> beträgt 0...∞. Eine Streamingquelle ist eindeutig über ihren Namen bestimmt. Die zu implementierenden Schnittstellen sind in Kapitel 4.2.3 definiert. In Abhängigkeit des zum Einsatz kommenden DevXML-Clients ist ein geeignetes Protokoll, das MP4 über HTTP Live Streaming, RV über RTSP, WMV über MMS und WMV über HTTP umfasst, für die Übertragung zu wählen. Die Anzeigemodalitäten Breite und Höhe werden über die Attribute *width* und *height* unter Angabe der Pixelanzahl bestimmt. Siehe hierzu Tabelle 31.

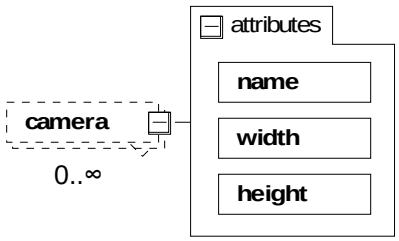
Diagramm		
Kardinalität	0...∞	
Kindelemente	-	
Attribute	erforderlich:	name, width, height

Tabelle 31 Darstellung des DevXML-Elements <*camera*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Für die nähere Charakterisierung eines Geräteteilzustandes integriert der DevXML-Sprachumfang das Instrument der Variablenassoziation. Jedem Platz kann unter Verwendung des Kindelements <*statevariable*> eine beliebige Anzahl von Variablen zugeordnet werden. Es wird hierbei die Existenz einer UPnP-Variablen, deren Änderungen über den Publisher-Subscribe-Mechanismus verfolgt werden können, vorausgesetzt. Die Namensgebung für die Referenz *name* folgt der Bildungsvorschrift

name = „@“ + [Name des UPnP-Service] + „:“ [Name der UPnP-Statusvariable]

Die im User Interface anzuzeigende Kurzbeschreibung wird über das Attribut *title* gesetzt. Das Attribut *unit* ist für die Angabe der mit einer Statusvariablen verbundenen physikalischen Einheit bestimmt. Die zur Anwendung kommende Kodierung lehnt an IEEE 1451 an. Ist der Bezug nicht das Gerät, von dem der DevXML-Code heruntergeladen wurde, so ist entweder der Unique Device Name oder die ergänzende Gerätebeschreibung gemäß Kapitel 4.2.1 anzugeben. Hierfür sind die Attribute *deviceUDN* repektive *deviceAddName* zu verwenden. Siehe Tabelle 32.

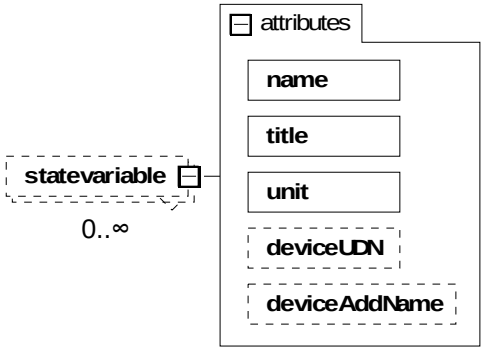
Diagramm		
Kardinalität	0...∞	
Kindelemente	-	
Attribute	erforderlich: optional:	name, title, unit deviceUDN, deviceAddName

Tabelle 32 Darstellung des DevXML-Elements <*statevariable*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Die Grundlage bildet ein aus 30 Byte bestehendes Feld. Dieses umfasst neben den 7 SI-Basiseinheiten m, kg, s, A, K, mol und cd 22 abgeleitete SI-Einheiten[135]. Gegenüber dem Standard IEEE 1451, der nur 9 unterschiedliche SI-Einheiten vorsieht[136], bietet der Sprachumfang von DevXML somit die Möglichkeit von weitaus differenzierteren Angaben.

[135] DIN: Einheiten und Begriffe für physikalische Größen, S1. ff. (DIN 1301, Teil 1).
[136] IEEE: IEEE Standard 1451.2-1997, S. 15.

Tabelle 33 Die Einheitenkodierung von Statusvariablen basiert auf dem Internationalen Einheitensystem.

Zeilenindex i	Größe	Einheit	Einheitenzeichen
1	dimensionslos	-	-
2	Länge	Meter	m
3	Masse	Kilogramm	kg
4	Zeit	Sekunde	s
5	Stromstärke	Ampere	A
6	Thermodynamische Temperatur	Kelvin	K
7	Stoffmenge	Mol	mol
8	Lichtstärke	Candela	cd
9	ebener Winkel	Radiant	rad
10	Raumwinkel	Steradiant	sr
11	Frequenz	Hertz	Hz
12	Kraft	Newton	N
13	Druck	Pascal	Pa
14	Energie, Arbeit	Joule	J
15	Leistung	Watt	W
16	elektrische Ladung	Coulomb	C
17	elektrische Spannung	Volt	V
18	elektrische Kapazität	Farad	F
19	elektrischer Widerstand	Ohm	Ω
20	elektrischer Leitwert	Siemens	S
21	magnetischer Fluss	Weber	Wb
22	magnetische Flussdichte	Tesla	T
23	Induktivität	Henry	H
24	Celsius-Temperatur	Grad Celsius	°C
25	Lichtstrom	Lumen	lm
26	Beleuchtungsstärke	Lux	lx
27	Radioaktivität	Becquerel	Bq
28	Energiedosis	Gray	Gy
29	Äquivalentdosis	Sievert	Sv
30	Katalytische Aktivität	Katal	kat

Der Exponent Exp_i einer SI-Basiseinheit bzw. einer abgeleiteten SI-Einheit E_i wird über den zugehörigen Feldeintrag F_i quantifiziert. Es gilt der Zusammenhang:

$$Exp_i = F_i - 128 \text{ mit } 2 \leq i \leq 30 \tag{4.2}$$

Seien a_k mit $1 \leqq k \leqq m$ all die Feldnummern mit $Exp_{ak} \neq 0$, dann setzt sich die Einheit E der zu beschreibenden Größe aus dem nachstehenden Produkt zusammen:

$$E = E_{a1}^{Exp_{a1}} \cdot E_{a2}^{Exp_{a2}} \cdot E_{a3}^{Exp_{a3}} \cdot (\cdots) E_{am}^{Exp_{am}} \tag{4.3}$$

Die Auswertung wird vorgenommen, sofern F_1 einen Wert ungleich 0 annimmt. Der initiale Wert aller F_i mit $1 \leqq i \leqq 30$ beträgt 128. Durch die Zuweisung $F_1 = 0$ wird die Einheit E als dimensionslos gekennzeichnet. In DevXML ist die Nomenklatur

$$unit = „[a_1] = F_{a1}; [a_2] = F_{a2}; \ldots ; [a_m] = F_{am}”$$

zu verwenden.

Die Einheit $\frac{m}{s^2}$ lässt sich demnach folgendermaßen darstellen:

unit = „[2] = 129; [4] = 126“

Die Kennzeichnung der Dimensionslosigkeit erfolgt mit:

unit = „[1] = 0“

Für Anwendungen mit der Erfordernis dynamischer Einheitenanpassung ist dem Attribut *unit* alternativ eine Variable vom Typ base64, die ein 30 Byte umfassendes Feld repräsentiert, zuordenbar.

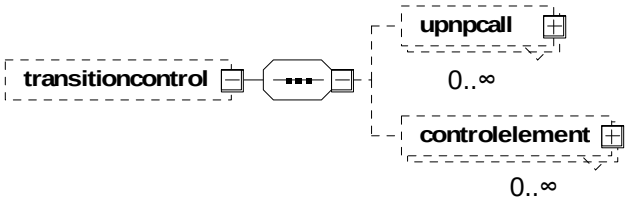
4.3.2.3 Elemente zur Transitionsspezifikation

Gemäß Kapitel 4.3.2.1 werden Gerätefunktionen, die Zustandsübergänge bewirken, als Transitionen modelliert. Diese bilden die Grundlage für die Interaktion mit dem Gerätenutzer. Das Element <*transitioncontrol*> mit seinen untergeordneten Kindelementen dient der Bereitstellung von Informationen für die Darstellung transitionsspezifischer User Interfaces. Der betreffende DevXML-Codeabschnitt ist auszuwerten, sobald vom User Interface eine im aktuellen Gerätezustand ausübbare Transition zur Anzeige gebracht werden soll. Die Modalitäten des Feuerns einer Transition sind über das Element <*transitionaction*> und zugehörige Subelemente zu spezifizieren.

Bei der Generierung von User-Interfaces können Ergebnisse von UPnP-Funktionsaufrufen sowie aktuelle Werte von Statusvariablen einbezogen werden. Zustandsübergänge selbst werden entweder durch Nutzerinteraktion oder automatischen Wertabgleich bewirkt. Diese Flexibilität wird realisiert, indem DevControl-Attributen nicht nur statische Werte, sondern auch Variablen zuweisbar sind.

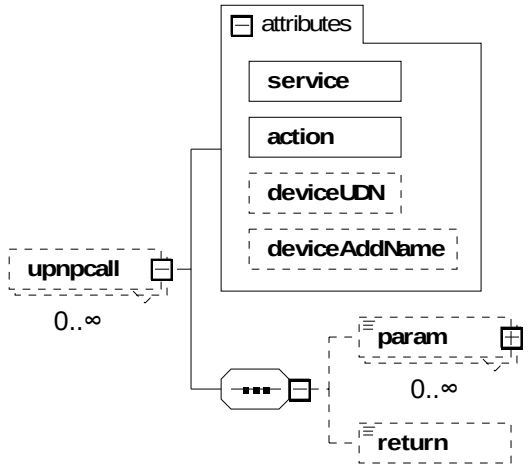
Das optionale Element <*transitioncontrol*> weist die Kardinalität 0...1 auf. Sofern eine Transition kein User Interface bedingt – zum Beispiel infolge einer rein zeitabhängigen Wirkweise – kann die Angabe unterbleiben. <*transitioncontrol*> beinhaltet die Kindelemente <*upnpcall*> und <*controlelement*>, mit denen Einfluss auf den funktionalen Ablauf und die grafische Gestaltung genommen werden können. Siehe hierzu Tabelle 34.

Tabelle 34 Darstellung des DevXML-Elements <*transitioncontrol*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Diagramm	
Kardinalität	0...1
Kindelemente	upnpcall, controlelement
Attribute	-

Mit dem Kindelement <*upnpcall*> werden UPnP-Funktionsaufrufe initiiert, deren Output-Parameter bei der User Interface-Generierung zu berücksichtigen sind. UPnP-Service und -Aktion werden über die Attribute *service* und *action* benannt. Soll ein externes – d. h. nicht das den DevXML-Code bereitstellende – Gerät referenziert werden, ist analog zum Element <*statevariable*> entweder das Attribut *deviceUDN* oder *deviceAddName* anzugeben. Für die Angabe von Parametern ist das untergeordnete Element <*param*> zu verwenden. Mit dem Element <*return*> kann der Rückgabewert einer Aktion einer DevXML-Variablen zugewiesen werden. Hierzu ist der Variablenname vom Element <*return*> zu umschließen. Tabelle 35 zeigt das Element <*upnpcall*> in der Übersicht.

Tabelle 35 Darstellung des DevXML-Elements <*upnpcall*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Diagramm		
Kardinalität	0...∞	
Kindelemente	param, return	
Attribute	erforderlich: optional:	service, action deviceUDN, deviceAddName

Ein Parameter ist über den Namen, den Typ und die mit ihm beabsichtigte Richtung des Informationsflusses bestimmt. In dieser Reihenfolge lauten die zum Element <*param*> gehörigen Attribute *name*, *type* und *dir*. Das Attribut *name* muss einen gültigen UPnP-Parameternamen bezeichnen. Dem Attribut *type* ist ein Parameter aus Tabelle 20 zuzuweisen. Inputparameter sind durch die Attributzuweisung *dir='in'*, Outputparameter durch die Attributzuweisung *dir='out'* zu kennzeichnen. Das Element <*param*> umschließt eine Zeichenkette, die alternativ für eine Variable oder einen statischen Wert steht.

Variablen werden mit dem Präfix „@“ eingeleitet und gliedern sich in die Klassen UPnP-Statusvariablen und Variablen, die innerhalb eines DevXML-Dokuments definiert werden (DevXML-Variablen). Die Nomenklatur zuerst genannter Klasse wurde bereits in Kapitel 4.3.2.2 „Platzspezifikation“ erläutert. Zuletzt genannte Klasse ist im Kontext einer Transition gültig und unterliegt der Einschränkung, nicht mit „@urn:“ beginnen zu dürfen. Hiervon abgesehen ist der auf das „@“-Zeichen folgende String frei wählbar. In Abhängigkeit des Attributs *dir* ergeben sich unterschiedliche Eingabemöglichkeiten. Gültige Input-Parameter umfassen Statusvariablen, DevXML-Variablen und statische Werte.

Für die Referenzierung von Variablen externer Geräte stehen wie beim Element <*state-variable*> die Attribute *deviceUDN* und *deviceAddName* zur Verfügung. Als Output-Parameter sind ausschließlich DevXML-Variablen zulässig. Tabelle 36 zeigt das Element <*param*> im Überblick.

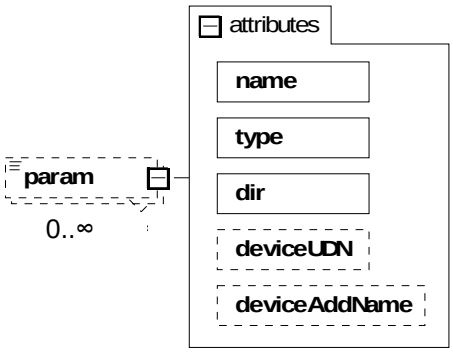
Diagramm		
Kardinalität	0...∞	
Kindelemente	-	
Attribute	erforderlich: optional:	name, type, dir deviceUDN, deviceAddName

Tabelle 36 Darstellung des DevXML-Elements <*param*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Mit <*controlelement*> wird ein Steuerelement des User Interface beschrieben. Mit dem Ziel der Referenzierbarkeit ist einem Steuerelement mittels *name*-Attribut ein innerhalb des <*transitioncontrol*>-Abschnittes eindeutiger Name zuzuordnen. Wird das Attribut *externalUI* des Elements <*transition*> auf *true* gesetzt, muss der Silverlight-Code ein gleichnamiges Steuerelement beinhalten. Der mit dem Attribut *type* spezifizierte Steuerelementtyp muss mit dem Typ des entsprechenden Silverlight-Steuerelements gemäß Tabelle 37 korrespondieren.

Steuerelementtyp in DevXML	**Steuerelementtyp in Silverlight**
table	DataGrid
label	TextBlock
textbox	TextBox
checkbox	CheckBox
slider	Slider
button	Button

Tabelle 37 Typenkorrespondenz bei Setzen des zum Element *transition* gehörigen Attributs *externalUI* auf *true*. Die Zuordnung ist eineindeutig.

Sofern das Attribut *externalUI* nicht den Wert *true* annimmt, werden alle Steuerelemente im Kontext der jeweiligen Transition autogeneriert. Jedem Steuerelement ist über das obligatorische Attribut *input* initial ein statischer Wert oder eine Variable zuzuweisen. Eignet sich ein Steuerelement nicht nur für die Anzeige, sondern auch für die Auswahl von Daten, ist das Selektionsergebnis unter Nutzung des Attributs *output* einer DevXML-Variablen zuweisbar.

Bei Verwendung des Steuerelementtyps *slider* wird der Wertebereich über die Attribute *min* und *max* vorgegeben. Der Verweis auf Statusvariablen externer Geräte ist mit den Attributen *deviceUDN* und *deviceAddName* möglich. In Tabelle 38 sind die wesentlichen Charakteristika von <*controlelement*> zusammengestellt.

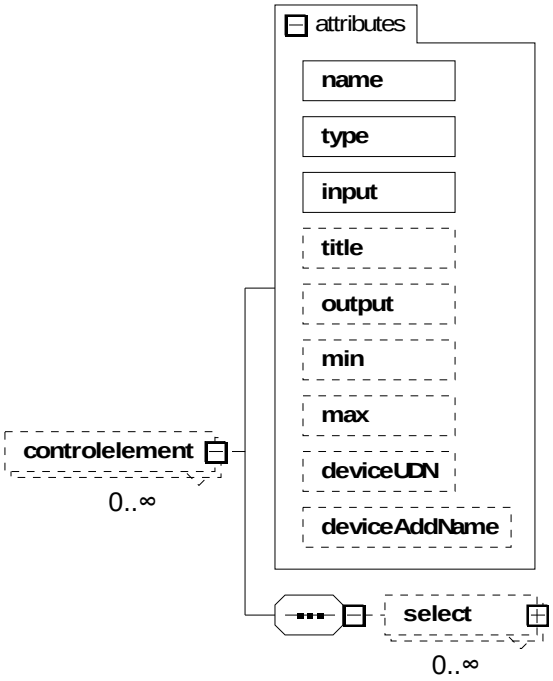
Diagramm		
		
Kardinalität	0...∞	
Kindelemente	select	
Attribute	erforderlich: optional:	name, type, input title, output, min, max, deviceUDN, deviceAddName

Tabelle 38 Darstellung des DevXML-Elements <*controlelement*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Zum Zwecke der Referenzierbarkeit ist jedem Kontrollelement über das *name*-Attribut ein transitionsweit eindeutiger Name zuzuordnen. Als *type*-Attribut können die in Tabelle 37 beschriebenen DevXML-Steuerelementtypen angegeben werden. Die Zuweisung eines Wertes erfolgt über das *input*-Attribut. Grundsätzlich zulässig sind Freitextangaben, UPnP-Statusvariablen und DevXML-Variablen. Bei Nutzung des Steuerelementtyps *table* sind Besonderheiten zu beachten. Für Vorgaben im Hinblick auf anzuzeigende Spalten und zu verwendende Variablen der aktuell ausgewählten Zellinhalte ist das Kindelement <*select*>, das exklusiv an den Steuerelementtyp *table* gebunden ist, einzusetzen. Das erforderliche Attribut *name* stellt den Bezug zu einer verfügbaren Spalte her. Zugleich ist über das Attribut *show* zu spezifizieren, ob die referenzierte Spalte im User Interface zu visualisieren ist oder nicht. Die Kodierung erfolgt über die booleschen Ausdrücke *true* und *false*. Das optionale Attribut *output* ermöglicht die Speicherung des Zellinhaltes, der durch die Spaltenreferenz und die getroffene Zeilenauswahl eindeutig bestimmt ist, in einer Variablen. Tabelle 39 stellt die Attribute des Elements zusammenfassend dar.

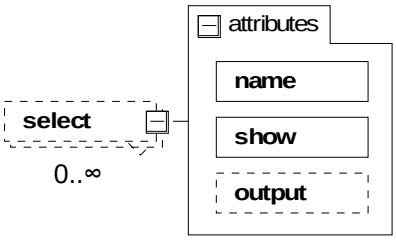
Diagramm		
Kardinalität	0...∞	
Kindelemente	-	
Attribute	erforderlich: optional:	name, show output

Tabelle 39 Darstellung des DevXML-Elements *<select>* mit Angaben zu Kardinalität, Kindelementen und Attributen.

Die Funktionsweise wird anhand von Quellcode 2 verdeutlicht.

Quellcode 2 Wirkweise des Elements *<select>*

```
<select name='c1' show='true' output='Var1'>
<select name='c2' show='true' output='Var2'>
<select name='c3' show='false' output='Var3'>
```

Tabelle 40 stellt die Assoziation der Zellinhalte mit den Variablen *Var1*, *Var2* und *Var3* heraus. Im User Interface werden ausschließlich die ersten beiden Spalten, bei denen das *show*-Attribut den Wert *true* annimmt, angezeigt.

c_1	c_2	c_3
r_{11}	r_{12}	r_{13}
Var1 = r_{21}	Var2 = r_{22}	Var3 = r_{23}
...	...	...
r_{n1}	r_{n2}	r_{n3}

Tabelle 40 Speicherung von Tabelleninhalten. Der Anzeigebereich ist hellblau hinterlegt. Er ist auf Variablen beschränkt, deren jeweiliges *show*-Attribut den Wahrheitswert *true* annimmt. Die zweite Zeile sei selektiert.

Sind die Inhalte einer nicht darzustellenden Spalte für die weitere Gerätesteuerung ohne Belang, kann die Angabe des Elements *<select>* hierfür unterbleiben. Wird das *type*-Attribut auf den Wert *label* gesetzt, ist über das *input*-Attribut ein String zuzuweisen. Dieser beschreibt den darzustellenden Text. Das *output*-Attribut wird in diesem Kontext aufgrund der ausschließlichen Anzeigefunktion eines Labels nicht verwendet.

Die Nutzung des DevXML-Steuerelementtyps *textbox* erfordert die initiale Zuweisung einer Zeichenkette über das *input*-Attribut. Die Speicherung des aktuellen Wertes einer Textbox in einer Variablen wird über das optionale *output*-Attribut ermöglicht. Analog wird mit den Steuerelementen *checkbox* und *slider* verfahren. Als Datentypen sind *boolean* bzw. *double* zu verwenden. Das *title*-Attribut, das eine Zeichenkette aufnimmt, dient hier ergänzend der Angabe eines Titels. Diese Zusatzinformation erscheint in unmittelbarer Umgebung eines *checkbox*- oder *slider*-Steuerelements.

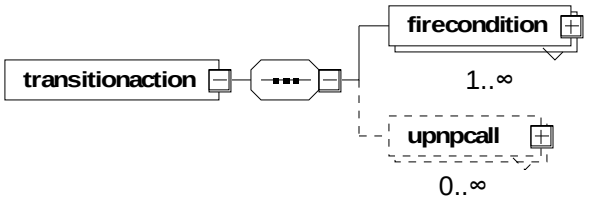
Der mit einem *button*-Steuerelement verbundene Textinhalt ist über das *input*-Attribut zu spezifizieren. Da Änderungen durch den Nutzer nicht möglich sind, kommt dem *output*-Attribut keine Funktion zu. Die möglichen Ausprägungen des Elements <*controlelement*> in Abhängigkeit des *type*-Attributs sind in Tabelle 41 zusammengefasst.

Tabelle 41 Ausprägungen des Elements <controlelement> in Abhängigkeit des type-Attributs.

type-Attribut	input-Attribut	Datentyp input-Attribut	output-Attribut	Datentyp output-Attribut	UI-Darstellung	Sonstige Attribute
table	X	proprietäres Datenformat	X	string (Eintrag der ausgewählten Zeile)	Anzeige/Auswahl	name
label	X	string	-	-	Anzeige	name
textbox	X	string	X	string	Anzeige/Auswahl	name
checkbox	X	boolean	X	boolean	Anzeige/Auswahl	name, title
slider	X	double	X	double	Anzeige/Auswahl	name, title, min, max
button	X	string	-	-	Anzeige	name

Das Element <*firecondition*> unterhalb des in Tabelle 42 dargestellten Zweiges <*transitionaction*> beschreibt, welche Ereignisse das Feuern einer Transition bewirken. Per Konvention ist die Überführung in den Folgezustand an die Bedingung geknüpft, dass die im Codesegment <*upnpcall*> aufgeführten Funktionen erfolgreich ausgeführt wurden. Andernfalls gilt die Transition als nicht gefeuert.

Tabelle 42 Darstellung des DevXML-Elements <*transitionaction*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Diagramm	
Kardinalität	1
Kindelemente	firecondition, upnpcall
Attribute	-

Es wird differenziert in Haupt- und Subtypen, die über die Attribute *type* und *subtype* näher qualifiziert werden. Mit der Attributzuweisung *type='UIEvent'* erfolgt die Verarbeitung von Ereignissen, die auf Steuerelemente zurückgehen. Ergänzend werden steuerelementspezifische Subtypen gemäß Tabelle 43 bereitgestellt. Somit ist zum Beispiel die Selektion einer Tabellenzeile, das Setzen einer Checkbox oder der Klick auf einen Button potentiell transitionsauslösend.

Tabelle 43 Gültige Subtypen eines UIEvent.

Steuerelement	Unterstützter Subtyp	Beschreibung
table	OnSelectionChanged	Selektion einer Tabellenzeile
label	-	-
textbox	OnContentChanged	Inhalt einer Textbox wurde modifiziert
checkbox	OnStateChanged	Haken einer Checkbox wurde gesetzt oder entfernt
slider	OnValueChanged	Stellung eines Schiebereglers wurde verändert
button	OnClick	Klick auf einen Button

Ein Steuerelement wird eindeutig referenziert, indem dessen Name dem Attribut *UIname* zugewiesen wird. Mit dem zweiten Haupttyp *VariableState* kann die Feuerung einer Transition in Abhängigkeit des Zustands einer Variablen modelliert werden. Der Variablenname, der über das Attribut *varname* anzugeben ist, bezieht sich wahlweise auf eine Status- oder DevXML-Variable und folgt den bereits beschriebenen Konventionen.

Das Attribut *vartype* kennzeichnet den zugrunde liegenden Datentyp, gültige Werte wurden weiter oben im Zusammenhang mit dem Element *<param>* diskutiert (siehe Tabelle 20). *Char*, *string* und *boolean* gestatten die Prüfung auf Gleichheit bzw. Ungleichheit zwischen einer Variablen und dem Wert des Attributs *varvalue*. Die entsprechenden Vergleichsoperatoren „=“ (gleich) und „!=“ (ungleich) sind über das Attribut *varoperator* festzulegen. Mit numerischen Datentypen (ui1, ui2 etc.) wird analog verfahren. Zusätzlich können die Vergleichsoperatoren „<“ (kleiner als) und „>“ (größer als) verwendet werden. Tabelle 44 gibt die Sachverhalte unter Nennung gültiger Eingaben für *varvalue* wieder.

Tabelle 44 Abhängigkeiten zwischen Attributen bei Verwendung des Haupttyps VariableState.

vartype	varoperator	varvalue
char	=, !=	Zeichenkette der Länge 1
string	=, !=	Zeichenkette beliebiger Länge
boolean	=, !=	Wahrheitswert true oder false
ui1, ui2, ui4, i1, i2, i4, int, r4, float, r8	=, !=, <, >	Wertebereich entsprechend Datentyp

Hinsichtlich der überblicksartigen Zusammenstellung relevanter Informationen zum Element <*firecondition*> sei auf Tabelle 45 verwiesen.

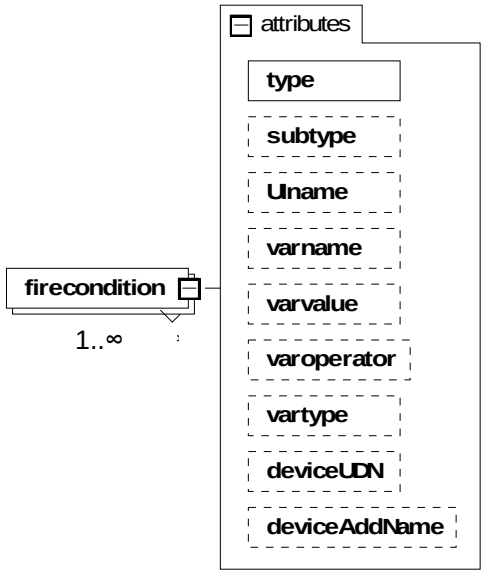
Diagramm		
Kardinalität	1...∞	
Kindelemente	-	
Attribute	erforderlich: optional:	type subtype, UIname, varname, varvalue, varoperator, vartype, deviceUDN, deviceAddName

Tabelle 45 Darstellung des DevXML-Elements <*firecondition*> mit Angaben zu Kardinalität, Kindelementen und Attributen.

Während das zu <*transitionaction*> gehörige Kindelement <*firecondition*> der Festlegung von Kriterien für Zustandsübergänge dient, werden mit dem Kindelement <*upnpcall*> die mit der Transitionsausübung verbundenen UPnP-Funktionsaufrufe bestimmt. Auf die Anwendung wurde im Zusammenhang mit dem Element <*transitioncontrol*> ausführlich eingegangen.

4.3.3 Navigation

Offensichtlich lässt sich mit Petrinetzen der Folgezustand nach Ausführung einer Transition bestimmen. Im Zusammenhang mit DevXML wird die formalisierte Darstellung des Weiteren für die geführte Navigation ausgenutzt. Die Führung besteht in der Ermittlung eines Pfades vom Ausgangs- zum Zielzustand. Der Zielzustand ergibt sich unmittelbar aus der Anwendervorgabe hinsichtlich der auszuführenden Transition. Für die Traversierung von Graphen mit Baumstruktur eignen sich die grundlegenden Verfahren Tiefen- und Breitensuche, deren Unterschiede im Folgenden erläutert werden. Seien F_k und F_{k+1} der k-te respektive der (k+1)-te Folgeknoten eines Ausgangsknotens. Die Tiefensuche ist dann dadurch gekennzeichnet, dass F_{k+1} erst aufgesucht wird, wenn alle Folgeknoten von F_k aufgesucht wurden. Jeder Folgeknoten wird hierbei selbst zum Ausgangsknoten, für dessen Unterbaumstruktur die beschriebene Traversierungsvorschrift in gleichem Maße gilt.

Bei der Breitensuche werden die direkten Folgeknoten eines Ausgangsknotens unmittelbar aufgesucht. Jeder Folgeknoten wird hierbei zu einer sequentiell abzuarbeitenden Liste von Ausgangsknoten hinzugefügt. Anhand der Baumstruktur in Abbildung 65 wird die Wirkweise exemplarisch dargestellt. Die Werte in Klammern geben die Reihenfolge der Knoten bei der Graphentraversierung an. Bei der Tiefensuche werden die Knoten K_0–K_8 in der Reihenfolge K_0, K_1, K_5, K_6, K_2, K_3, K_7, K_8 und K_4 aufgesucht. Die hiervon abweichende Reihenfolge K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7 und K_8 ergibt sich bei der Breitensuche.

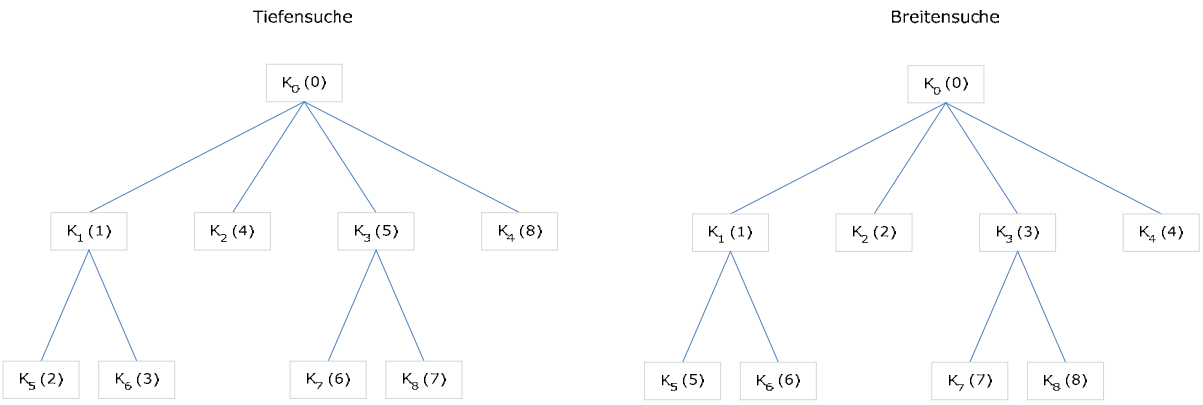


Abbildung 65 Graphentraversierung bei der Tiefen- und Breitensuche. Die Werte in Klammern geben die Reihenfolge, in der die Knoten durchsucht werden, wieder. Für die Tiefensuche ergibt sich die Reihenfolge K_0, K_1, K_5, K_6, K_2, K_3, K_7, K_8 und K_4. Die Breitensuche führt zu K_0, K_1, K_2, K_3, K_4, K_5, K_6, K_7 und K_8.

Die skizzierten graphenalgorithmischen Verfahren können in modifizierter Form für die Bestimmung der Erreichbarkeitsmenge $E(M_0)$ von Petrinetzstrukturen gemäß Definition 16 angewendet werden. Die Kernidee besteht in der Herstellung einer Äquivalenzbeziehung zwischen Knoten und Markierung. Es gilt einschränkend, dass bei der Traversierung nur erstmalig aufgesuchte Markierungen als Knoten interpretiert werden. Die Beziehung zwischen Knoten und Markierung ist somit eineindeutig. Diese Eigenschaft bleibt auch dann erhalten, wenn eine Markierung über mehrere Pfade erreichbar ist. Es folgt, dass die Tiefensuche-Erreichbarkeitsanalyse und Breitensuche-Erreichbarkeitsanalyse im Allgemeinen unterschiedlichen Baumstrukturen entsprechen. In den folgenden Ausführungen wird die Breitensuche-Erreichbarkeitsanalyse näher behandelt.

Neben der Erreichbarkeitsmenge $E(M_0)$ wird mit Algorithmus 1 die im weiteren Verlauf für die Bestimmung des kürzesten Pfades benötigte Adjazenzmatrix A und die zugehörige Matrix A' ermittelt. Die Adjazenzmatrix A weist die folgende Form auf:

$$A = \begin{array}{c} \begin{matrix} M_0 & M_1 & & M_n \end{matrix} \\ \begin{pmatrix} A_{M0,M0} & A_{M0,M1} & \cdots & A_{M0,Mn} \\ A_{M1,M0} & A_{M1,M1} & \cdots & A_{M1,Mn} \\ \vdots & \vdots & \ddots & \vdots \\ A_{Mn,M0} & A_{Mn,M1} & \cdots & A_{Mn,Mn} \end{pmatrix} \end{array} \begin{matrix} M_0 \\ M_1 \\ \\ M_n \end{matrix} \quad \text{mit} \quad (4.4)$$

$$A_{Mx,My} = \begin{cases} 0 \text{ wenn } M_y \text{ keine direkte Folgemarkierung von } M_x \text{ ist} \\ 1 \text{ wenn } M_y \text{ direkte Folgemarkierierung von } M_x \text{ ist} \end{cases} \quad \text{für}$$

$0 \le x \le n$ und $0 \le y \le n$

Die Matrix A' beinhalte für jedes Paar von Markierungen aus A die den Übergang bewirkende Transition t. Sie weist dann folgende Struktur auf:

$$A' = \begin{array}{c} \begin{matrix} M_0 & M_1 & & M_n \end{matrix} \\ \begin{pmatrix} A'_{M0,M0} & A'_{M0,M1} & \cdots & A'_{M0,Mn} \\ A'_{M1,M0} & A'_{M1,M1} & \cdots & A'_{M1,Mn} \\ \vdots & \vdots & \ddots & \vdots \\ A'_{Mn,M0} & A'_{Mn,M1} & \cdots & A'_{Mn,Mn} \end{pmatrix} \end{array} \begin{matrix} M_0 \\ M_1 \\ \\ M_n \end{matrix} \quad \text{mit} \quad (4.5)$$

$$A'_{Mx,My} = \begin{cases} 0 \text{ wenn } \neg(\exists t \in T) : M_x[t > M_y \\ t \text{ wenn } \exists t \in T : M_x[t > M_y \end{cases} \quad \text{für}$$

$0 \le x \le n$ und $0 \le y \le n$

Die Matrix A' ist Bestandteil der modifizierten Form der Breitensuche-Erreichbarkeitsanalyse. Siehe hierzu Algorithmus 1. Für jedes Petrinetz mit endlicher Erreichbarkeitsmenge terminiert Algorithmus 1. Die Eigenschaft der Endlichkeit trifft auf jedes k-beschränkte Petrinetz zu[137]. Vergleiche auch **Fehler! Verweisquelle konnte nicht gefunden werden.**.

[137] Priese/Wimmel: Petri-Netze, S. 24.

Algorithmus 1 Modifizierte Form der Breitensuche-Erreichbarkeitsanalyse

Schritt 1

Initialisierung der Variablen aktueller Markierungsindex i, maximaler Markierungsindex max, Adjazenzmatrix A, Matrix A' und aktuelle Markierung M_i:

$i = 0$, $max = 0$, $A = 0$, $A' = 0$, $M_i = M_0$

Schritt 2

Für jede Transition t aus T:

Vorausgesetzt t ist aus M_i feuerbar, dann sind im Weiteren 2 Fälle zu unterscheiden:

Fall 1: $M_i[t$ ist bereits erfasst
Es gilt dann $M_i[t>M_j$ mit $0 \leqq j \leqq max$.

Fall 2: $M_i[t$ ist noch nicht erfasst
Die neue Markierung ist aufzunehmen:
$max = max + 1$, $j = max$, $M_{max} = M_i[t$

In beiden Fällen ist in der Adjazenzmatrix zu vermerken, dass ein direkter Weg von M_i nach M_j führt: $A[M_i,M_j] = 1$, $A'[M_i,M_j] = t$

Schritt 3

Es wird überprüft, ob die Breitenanalyse abgeschlossen ist. Zwei Fälle werden betrachtet:

Fall 1: $i = max$
Beendigung des Algorithmus.

Fall 2: $i \neq max$
Inkrementieren der Variable i:
$i = i + 1$
Weiter mit Schritt 2.

Exemplarisch wird die Anwendung von Algorithmus 1 auf das in Abbildung 66 dargestellte Petrinetz, dessen Struktur mit der des in Kapitel 6.6 behandelten Jukebox-Petrinetzes übereinstimmt, näher ausgeführt. An dieser Stelle erfolgt durch die Wahl der abweichenden Platzbezeichnungen P_0–P_5 und Transitionsbezeichnungen t_1–t_8 bewusst eine Reduktion auf eine stark formalisierte Beschreibungsebene, um so die Eigenschaften des Algorithmus in den Mittelpunkt zu rücken.

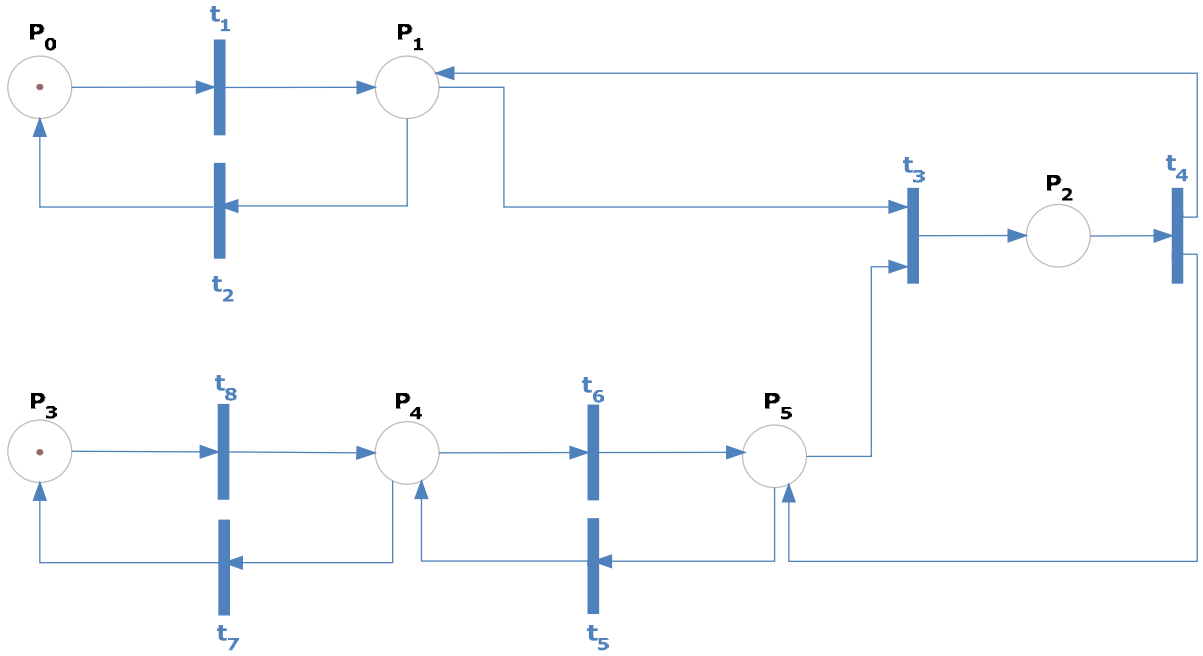


Abbildung 66 Beispielnetz für die Breitensuche-Erreichbarkeitsanalyse. Strukturell stimmt dieses mit dem Petrinetz der in Kapitel 6.6 vorgestellten Jukebox überein. Ausgehend von der Anfangsmarkierung $M_0 = (1\ 0\ 0\ 1\ 0\ 0)^T$ wird die Erreichbarkeitsmenge $E(M_0)$ bestimmt.

Die aus den Elementen M_0–M_6 bestehende Erreichbarkeitsmenge $E(M_0)$ wird ausgehend von $M_0 = (1\ \ 0\ \ 0\ \ 1\ \ 0\ \ 0)^T$ aufgebaut. Tabelle 46 stellt die Platzbelegungen der möglichen Markierungen im Überblick dar.

Tabelle 46 Erreichbarkeitsmenge $E(M_0)$ des Beispielnetzes. Insgesamt können sieben unterschiedliche Markierungen eingenommen werden.

	P0	**P1**	**P2**	**P3**	**P4**	**P5**
M0	1	0	0	1	0	0
M1	0	1	0	1	0	0
M2	1	0	0	0	1	0
M3	0	1	0	0	1	0
M4	1	0	0	0	0	1
M5	0	1	0	0	0	1
M6	0	0	1	0	0	0

Es folgt die Darstellung der zugehörigen Matrizen A und A'. Diesen ist z. B. zu entnehmen, dass durch Feuern der Transition t_6 aus M_2 heraus die Markierung M_4 eingenommen wird (siehe grau hinterlegte Felder).

$$A = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix}$$

$$A' = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & t_1 & t_8 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 0 & t_8 & 0 & 0 & 0 \\ t_7 & 0 & 0 & t_1 & t_6 & 0 & 0 \\ 0 & t_7 & t_2 & 0 & 0 & t_6 & 0 \\ 0 & 0 & t_5 & 0 & 0 & t_1 & 0 \\ 0 & 0 & 0 & t_5 & t_2 & 0 & t_3 \\ 0 & 0 & 0 & 0 & 0 & t_4 & 0 \end{pmatrix} \end{array} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix}$$

Die dem Algorithmusablauf entsprechende Baumstruktur zeigt Abbildung 67. Die in Klammern angegebenen Indizierungen spiegeln die Entstehung der Baumstruktur in ihrer zeitlichen Abfolge – und damit die Erfassungsrangfolge der Elemente aus $E(M_0)$ – wider. Zum wiederholten Male auftretende – folglich nicht als Knoten zu interpretierende – Folgemarkierungen sind als schraffierte Rechtecke abgebildet.

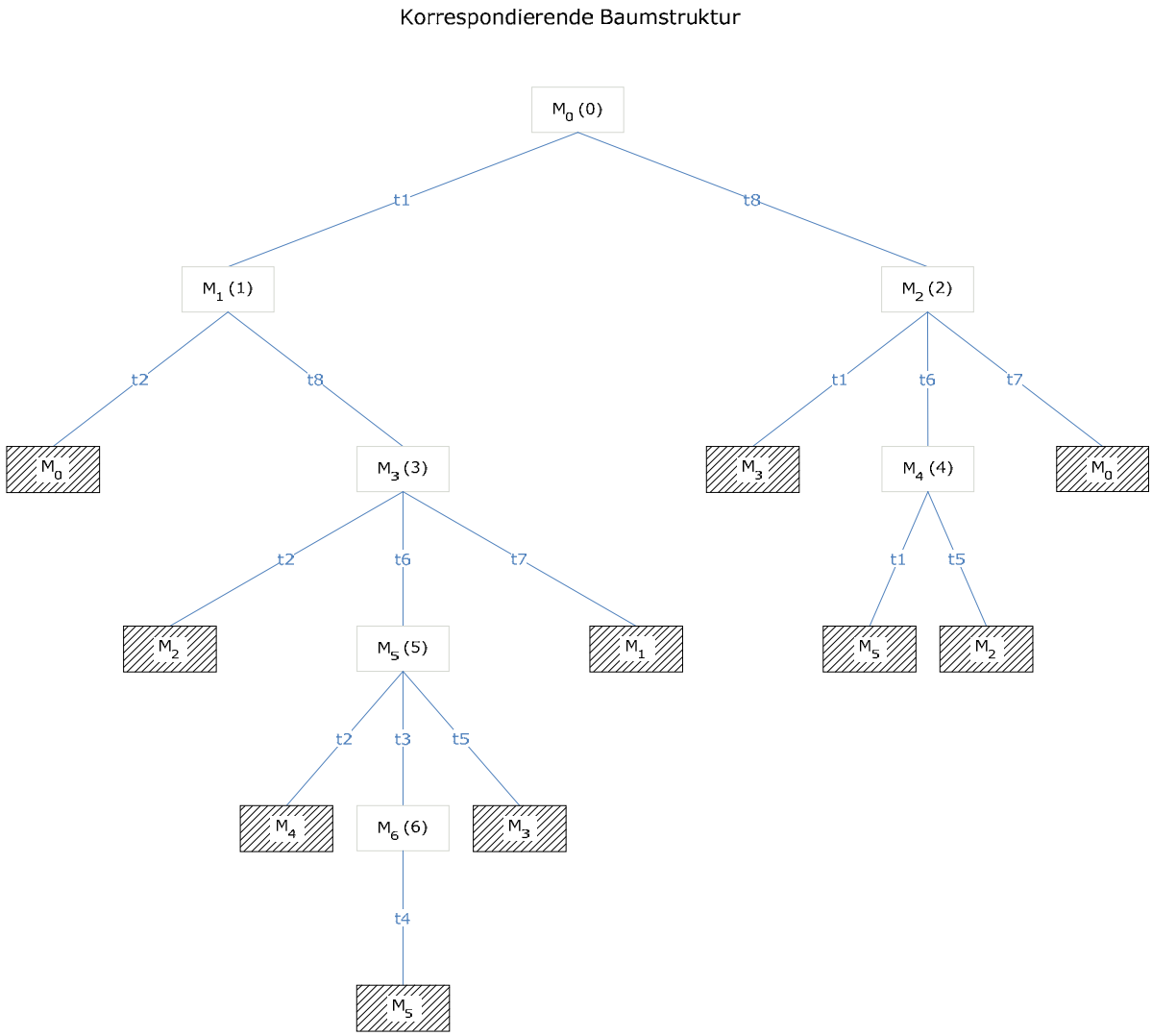


Abbildung 67 Zum Beispielnetz gehörige Baumstruktur. Aus den Werten in Klammern geht die zeitliche Abfolge dessen Entstehung bei der Breitensuche hervor. Auf der Verbindungslinie zwischen zwei Markierungen ist die Transition angegeben, die den Übergang bewirkt.

Eine im Rahmen der Gerätesteuerung durch den Nutzer getroffene Zielvorgabe hinsichtlich des anzustrebenden Gerätezustandes entspricht netztheoretisch dem Erreichen einer vorgegebenen Netzmarkierung M_l aus M_k heraus (mit $k < |E(M_0)|$ und $l < |E(M_0)|$). Dies setzt die Ermittlung eines gültigen Pfades voraus. Mit der Maßgabe, die Anzahl erforderlicher Benutzerinteraktionen möglichst gering zu halten, wird bei den weiteren Betrachtungen eine minimale Pfadlänge gefordert. Die Modellierung erfolgt als ungewichteter Graph, jede gerichtete Kante zwischen zwei adjazenten Markierungen besitzt somit die Länge 1.

Mit dem Algorithmus von Floyd steht ein Verfahren zur Verfügung, das ausgehend von der Adjazenzmatrix A der Erreichbarkeitsmenge $E(M_0)$ die Kodierung der transitiven Hülle benachbarter Markierungen vornimmt[138]. Grundlage der Berechnungen sind grapheninhärente, transitive Beziehungen: Führt ein Weg von A nach B und ein Weg von B nach C, so führt auch ein Weg von A nach C. Der Algorithmus von Floyd ist dadurch gekennzeichnet, dass die Generierung der transitiven Hülle nach einmaligem Durchlaufen der Adjazenzmatrix abgeschlossen ist. Hierzu wird die Menge der potentiell als Zwischenstationen fungierenden Markierungen sukzessive erweitert. Die zuletzt hinzugefügte Zwischenstation wird über den Index y qualifiziert. Dieser durchläuft den Wertebereich von 0 bis $|E(M_0)|$-1. Somit ist mit der letzten Inkrementierung von y jede Markierung eine potentielle Zwischenstation.

Der kürzeste Weg von M_x nach M_j ist dann entweder der kürzeste Weg von M_x nach M_j unter ausschließlicher Verwendung der Zwischenstationen M_0–M_{y-1} (Pfad I) oder der kürzeste Weg von M_x nach M_y plus den Weg von M_y nach M_j unter ausschließlicher Verwendung der Zwischenstationen M_0–M_{y-1} auf den Teilabschnitten (Pfad II). Die Illustration dieses Sachverhalts ist Abbildung 68 zu entnehmen. Im eingezeichneten Fall weist der Pfad II, der über die neu hinzugefügte Markierung M_y führt, eine kürzere Distanz als Pfad I auf. Pfad II stellt somit mit einer Länge von 4 die kürzeste Entfernung von M_x nach M_j dar.

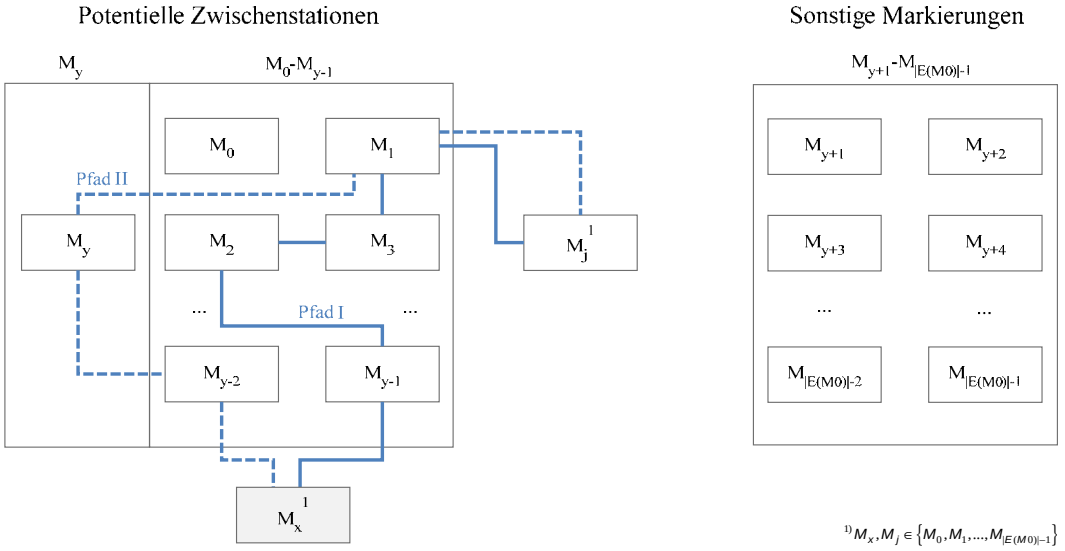


Abbildung 68 Bestimmung der minimalen Pfadlänge mit dem Algorithmus von Floyd. Die Menge der potentiellen Zwischenstationen wird sukzessive erweitert. Das zuletzt hinzugefügte Element trägt die Bezeichnung M_y. Der kürzeste Weg von M_x nach M_j für alle x,j aus 0, 1, ..., $|E(M_0)|$-1 ist dann entweder der kürzeste Weg von M_x nach M_j unter ausschließlicher Verwendung der Zwischenstationen M_0–M_{y-1} oder der kürzeste Weg von M_x nach M_y plus den kürzesten Weg von M_y nach M_j unter ausschließlicher Verwendung der Zwischenstationen M_0–M_{y-1} auf den Teilabschnitten.

[138] Sedgewick: Algorithmen, S. 541 ff.

Die Pfadlänge wird in der Matrix T festgehalten. In der modifizierten Form des Algorithmus wird darüber hinaus die Matrix T' erstellt, die alle zugehörigen Transitionsfolgen beinhaltet. Es besteht prinzipiell die Möglichkeit, dass zwischen zwei Markierungen mehrere Pfade mit minimaler Länge existieren. In diesem Fall ist die erste vom Algorithmus gefundene Lösung leitend. Dies führt zu Algorithmus 2.

Algorithmus 2 Modifizierte Form des Algorithmus von Floyd

Schritt 1

Initialisierung der Variablen transitive Hülle T und Matrix T' mit den Ergebnismatrizen A und A' der Breitensuche-Erreichbarkeitsanalyse:

T = A, T' = A'

Schritt 2

```
FOR y = 0 TO |E(M0)|-1
   FOR x = 0 TO |E(M0)|-1
      IF T[Mx,My] > 0                                              (Bed. 1)
         FOR j = 0 to |E(M0)|-1
            IF T[My,Mj] > 0                                        (Bed. 2)
               IF ((T[Mx,Mj] = 0) OR (T[Mx,My] + T[My,Mj] < T[Mx,Mj]))   (Bed. 3)
                     T[Mx,Mj] = T[Mx,My] + T[My,Mj]
                     T'[Mx,Mj] = T'[Mx,My] + T'[My,Mj]
```

Ein konkreter Anwendungsfall wird unter Nutzung der für das Beispielnetz in Abbildung 66 errechneten Matrizen A und A' erläutert. Zur Referenzierung des Interimszustandes der transitiven Hülle T zum Zeitpunkt der Initialisierung bzw. Inkrementierung von y wird dieser mit T_y bezeichnet. Selbiges gilt für die zugehörige Matrix T'. Mit $y = 0$ ergibt sich die erste Zwischenstation M_y zu M_0 und der erste Interimszustand der Matrix T zu T_0. Siehe hierzu nachfolgende Darstellungen.

Für alle Markierungen, die zu der als Zwischenstation hinzugefügten Markierung M_0 hinführen (Bed. 1), wird untersucht, ob sich unter Verwendung von M_0 neue bzw. kürzere Wege (Bed. 3) zu den von hier erreichbaren Markierungen (Bed. 2) ergeben. Im vorliegenden Beispiel führt der Vergleich der Zeile der Zwischenstation M_0 (= Zeile 0) mit den Zeilen der Startmarkierungen M_1 (= Zeile 1) und M_2 (= Zeile 2) zu neuen Pfaden. Die Zeile der Zwischenstation ist von einem Rechteck mit durchgezogener Linie umgeben. Für die Zeilen der Startmarkierungen wurden Rechtecke mit gestrichelten Linien gewählt. Für $M_y = M_1 ... M_6$ wird analog verfahren. Vergleiche hierzu den ebenfalls abgebildeten Fall $M_y = M_1$. Die Matrizen T_1 und T_1' beinhalten die im ersten Durchlauf ermittelten kürzesten Wege. Zu nennen sei beispielsweise der Eintrag $T_1[M_1,M_2] = 2$ mit der Transitionsfolge $T_1'[M_1,M_2] = (t_2;t_8)$. Dies entspricht $M_1[t_2>M_0[t_8>M_2$.

Der Algorithmus terminiert, nachdem die Berechnungsschritte auf die Matrizen T_6 und T_6' – unter Beibehaltung obiger Notation gilt hier $M_y = M_6$ – angewendet wurden. Die sich ergebenden Matrizen tragen nachfolgend die Bezeichnungen T_F und T_F'. Es ist zum Beispiel ersichtlich, dass der kürzeste Pfad von M_0 nach M_6 die Länge 4 aufweist und dieser mit der Transitionsfolge t_1, t_8, t_6, t_3 korreliert (siehe blau hervorgehobene Matrixeinträge). Dies entspricht der alternativen, an Definition 15 anlehnenden Schreibweise $M_0[t_1>M_1[t_8>M_3[t_6>M_5[t_3>M_6$.

M_y=M_0:

$$T_0 = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

$$T_0' = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & t_1 & t_8 & 0 & 0 & 0 & 0 \\ t_2 & 0 & 0 & t_8 & 0 & 0 & 0 \\ t_7 & 0 & 0 & t_1 & t_6 & 0 & 0 \\ 0 & t_7 & t_2 & 0 & 0 & t_6 & 0 \\ 0 & 0 & t_5 & 0 & 0 & t_1 & 0 \\ 0 & 0 & 0 & t_5 & t_2 & 0 & t_3 \\ 0 & 0 & 0 & 0 & 0 & t_4 & 0 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

M_y=M_1:

$$T_1 = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 2 & 2 & 1 & 0 & 0 & 0 \\ 1 & 2 & 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

$$T_1' = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 0 & t_1 & t_8 & 0 & 0 & 0 & 0 \\ t_2 & t_2;t_1 & t_2;t_8 & t_8 & 0 & 0 & 0 \\ t_7 & t_7;t_1 & t_7;t_8 & t_1 & t_6 & 0 & 0 \\ 0 & t_7 & t_2 & 0 & 0 & t_6 & 0 \\ 0 & 0 & t_5 & 0 & 0 & t_1 & 0 \\ 0 & 0 & 0 & t_5 & t_2 & 0 & t_3 \\ 0 & 0 & 0 & 0 & 0 & t_4 & 0 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

$$T_F = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} 2 & 1 & 1 & 2 & 2 & 3 & 4 \\ 1 & 2 & 2 & 1 & 3 & 2 & 3 \\ 1 & 2 & 2 & 1 & 1 & 2 & 3 \\ 2 & 1 & 1 & 2 & 2 & 1 & 2 \\ 2 & 3 & 1 & 2 & 2 & 1 & 2 \\ 3 & 2 & 2 & 1 & 1 & 2 & 1 \\ 4 & 3 & 3 & 2 & 2 & 1 & 2 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

$$T_F' = \begin{array}{c} \begin{matrix} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 \end{matrix} \\ \begin{pmatrix} t_1;t_2 & t_1 & t_8 & t_1;t_8 & t_8;t_6 & t_1;t_8;t_6 & t_1;t_8;t_6;t_3 \\ t_2 & t_2;t_1 & t_2;t_8 & t_8 & t_2;t_8;t_6 & t_8;t_6 & t_8;t_6;t_3 \\ t_7 & t_7;t_1 & t_7;t_8 & t_1 & t_6 & t_1;t_6 & t_1;t_6;t_3 \\ t_7;t_2 & t_7 & t_2 & t_7;t_8 & t_2;t_6 & t_6 & t_6;t_3 \\ t_5;t_7 & t_5;t_7;t_1 & t_5 & t_5;t_1 & t_5;t_6 & t_1 & t_1;t_3 \\ t_5;t_7;t_2 & t_5;t_7 & t_5;t_2 & t_5 & t_2 & t_5;t_6 & t_3 \\ t_4;t_5;t_7;t_2 & t_4;t_5;t_7 & t_4;t_5;t_2 & t_4;t_5 & t_4;t_2 & t_4 & t_4;t_3 \end{pmatrix} \begin{matrix} M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \end{matrix} \end{array} \quad Mx\downarrow$$

4.3.4 Modellzuverlässigkeit

Wie in Kapitel 4.3.2 „Sprachelemente“ erläutert, wird mit dem DevXML-Code eines Gerätes dessen zugehöriges Petrinetzmodell beschrieben. Die eingenommene Netzmarkierung repräsentiert hierbei den aktuellen Gerätezustand. Das Feuern von Transitionen sowie der Aufruf assoziierter Funktion werden zentral vom DevControl-Gateway vollzogen. Ebenso ermittelt dieser bei Zustandsübergängen die neue Netzmarkierung und leitet sie an das jeweilige Gerät unter Rückgriff auf die Methode *SetPlaceState(i4 PlaceID_X, i4 PlaceID_Y, i4 PlaceState)* weiter.

Im Folgenden wird auf die in DevControl bereitstehenden Mechanismen zur Abwendung ungültiger Zustände eingegangen. Jederzeit möglich ist offensichtlich die Überprüfung, ob die vom DevControl-Gateway übermittelte Netzmarkierung in der Erreichbarkeitsmenge $E(M_0)$ enthalten ist. Bei einem umfangreichen Netz und einem gleichzeitig ressourcenarmen Gerät kann dieses Vorgehen zu extremen Performanceeinbußen führen. Alternativ ist eine Validitätsprüfung anhand platzspezifisch gewichteter Markensummen durchführbar. Die zugrunde liegende Berechnungsmethodik wurde bereits in Kapitel 3.3 „Invariantenkalkül“ dargelegt. Voraussetzung hierfür ist die Kenntnis einer P-Invariante, die den zu berücksichtigenden Netzbereich überdeckt. Mit dem Ziel der Überdeckung ausschließlich sicherheitskritischer Bereiche kann die Auswahl der P-Invariante selektiv erfolgen[139]. Im Rahmen des hier vorgestellten Algorithmus werden ausschließlich nichtnegative P-Invarianten betrachtet. Daher sind im Weiteren die Komponenten von P-Invarianten implizit immer größer oder gleich Null.

Der im Unterkapitel 4.3.4.1 eingeführte Algorithmus von Martinez schafft die Voraussetzung für die formalisierte Bestimmung von sogenannten minimalen Support P-Invarianten. Es wird gezeigt, wie unter Verwendung der generierten Ergebnismenge P-Invarianten mit einer maximalen Anzahl besetzter Plätze ermittelt werden können. Als Hinführung folgen zunächst Definitionen zu den verständnisrelevanten Begriffen Support, minimale Support P-Invariante sowie minimale P-Invariante.

Definition 24

Der Support Supp(ϕ_{P1}) einer P-Invariante ϕ_{P1} stellt die Menge der Plätze dar, deren zugehörige Komponenten einen Wert größer Null annehmen. Der Support Supp(ϕ_{P1}) ist minimal, wenn er keinen weiteren Support außer sich selbst und der leeren Menge beinhaltet. Es existiert dann also kein Supp(ϕ_{P2}) mit:

$$\text{Supp}(\boldsymbol{\phi}_{P1}) \subset \text{Supp}(\boldsymbol{\phi}_{P2})$$

Definition 25

Eine P-Invariante ϕ_P, deren Support Supp(ϕ_P) minimal ist, wird als minimale Support P-Invariante $\phi_{P,min,Supp}$ bezeichnet.

[139] König/Quäck: Petri-Netze in der Steuerungs- und Digitaltechnik, S. 49 f.

Definition 26
Eine P-Invariante ϕ_{P1} wird als minimale P-Invariante bezeichnet, wenn keine P-Invariante ϕ_{P2} existiert mit:

$$\phi_{P1} \gneqq \phi_{P2}$$

(Für jedes i gilt die Beziehung $\phi_{P1}(i) \geqq \phi_{P2}(i)$ und es existiert mindestens ein i mit $\phi_{P1}(i) > \phi_{P2}(i)$)

Satz 6
Wird eine minimale Support P-Invariante $\phi_{P,min,Supp}$ durch den größten gemeinsamen Teiler ihrer Komponenten α geteilt, entsteht eine minimale P-Invariante $\phi_{P,min}$[140]. Es gilt dann:

$$\phi_{P,min} = \phi_{P,min,Supp} / \alpha$$

Satz 7
Jede P-Invariante lässt sich als eine Linearkombination mit positiven Koeffizienten aus minimalen Support P-Invarianten bestimmen[141].

Es ist zu beachten, dass nicht jede minimale P-Invariante auch eine minimale Support P-Invariante ist. Ein Netz N besitze die zwei minimalen Support P-Invarianten $\phi_{P1} = (2,1,0)$ und $\phi_{P2} = (0,1,2)$, die nach Satz 6 auch minimale P-Invarianten sind. Die Division der Linearkombination $\phi_{P1} + \phi_{P2} = (2,2,2)$ durch 2 ergibt die minimale P-Invariante $\phi_{P3} = (1,1,1)$. Jede einzelne der hier geltenden Beziehungen $Supp(\phi_{P1}) = \{s_1,s_2\} \subset Supp(\phi_{P3}) = \{s_1,s_2,s_3\}$ und $Supp(\phi_{P2}) = \{s_2,s_3\} \subset Supp(\phi_{P3}) = \{s_1,s_2,s_3\}$ ist gemäß Definition 24 hinreichende Bedingung dafür, dass ϕ_{P3} keine minimale Support P-Invariante darstellt.

4.3.4.1 Algorithmus von Martinez

Mit dem Algorithmus von Martinez wird für jede zum Petrinetz N gehörige P-Invariante mit minimalem Support mindestens eine zugehörige minimale Support P-Invariante ermittelt[142]. Entgegen den Ausführungen im Paper von Martinez und Silva umfasst die Ergebnismatrix nicht alle minimalen Support P-Invarianten[143]. Denn ausgehend von einer minimalen Support P-Invariante lassen sich unendlich viele weitere Support P-Invarianten ermitteln. Sei $\phi_{P1} = (2,1,0)$ eine minimale Support P-Invariante mit $Supp(\phi_{P1}) = \{s_1,s_2\}$, dann ist $\phi_{Pk} = k\phi_{P1}$ mit $Supp(\phi_{Pk}) = \{s_1,s_2\}$ und $k \in \mathbb{Z}$ ebenfalls eine minimale Support P-Invariante. Die vom Algorithmus ermittelten minimalen Support P-Invarianten bilden nach Satz 7 ein erzeugendes System für alle P-Invarianten mit nichtnegativen Komponenten. Der Algorithmus von Martinez ist nachfolgend angegeben.

[140] David/Alla: Discrete, Continuous, and Hybrid Petri Nets, S. 47 f.
[141] Martinez/Silva: A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net, S. 303.
[142] Martinez/Silva: A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net.
[143] David/Alla: Discrete, Continuous, and Hybrid Petri Nets, S. 50.

Algorithmus 3 *Algorithmus von Martinez*

Schritt 1

Initialisierung der Matrizen A und D: A = I, D = E_{AP}

Schritt 2

FOR j = 1 TO A_T

Schritt 2.1

Für jedes Zeilenpaar aus [D | A], für das nichtnegative Linearkombinationen mit dem Wert 0 in der j-ten Spalte von A existieren, ist genau eine Linearkombination mit dieser Eigenschaft der Matrix [D | A] hinzuzufügen.

Schritt 2.2

Zeilen, in denen die j-te Spalte einen von 0 verschiedenen Wert annimmt, sind zu streichen.

mit

A_P = Anzahl Plätze
A_T = Anzahl Transitionen
E_{AP} = A_P-te Einheitsmatrix
I = Inzidenzmatrix des Netzes N

Das Beispiel aus dem Abschnitt „Invariantenkalkül“ (siehe Kapitel 3.3) aufgreifend wird im Folgenden die Anwendung des Algorithmus von Martinez demonstriert. Die Platzanzahl $A_P = 5$ (Plätze s_1 bis s_5), die Transitionszahl $A_T = 3$ (Transitionen t_1 bis t_3) und die Inzidenzmatrix

$$I = \begin{pmatrix} -2 & 0 & 2 \\ -1 & 0 & 1 \\ 1 & -1 & -1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

führen zu Tabelle 47. Es werden die vier P-Invarianten $\Theta_{P1} = (1\ 0\ 2\ 2\ 0)^T$, $\Theta_{P2} = (0\ 1\ 1\ 1\ 0)^T$, $\Theta_{P3} = (1\ 0\ 2\ 0\ 2)^T$ und $\Theta_{P4} = (0\ 1\ 1\ 0\ 1)^T$ ausgewiesen. Im Vergleich zur Lösung gemäß Gleichung (3.43) ist zusätzlich die linear abhängige P-Invariante Θ_{P1} ein Vektor des erzeugenden Systems. Dies liegt darin begründet, dass nichtnegative Linearkombinationen der Ergebnisvektoren des Algorithmus von Martinez den Lösungsraum aufspannen. Damit ist auch Θ_{P1} erforderlich, denn es gilt $\Theta_{P1} = 2\Theta_{P1} + \Theta_{P3} - 2\Theta_{P4}$.

Tabelle 47 Anwendung des Algorithmus von Martinez. Die P-Invarianten der Ergebnismenge sind hellblau hinterlegt.

	i	j	D					A			Support der i-ten Zeile aus D	Gestrichen bei j	Anmerkung
Z_1	1	-	1	0	0	0	0	-2	0	2	$\{s_1\}$	1	-
Z_2	2	-	0	1	0	0	0	-1	0	1	$\{s_2\}$	1	-
Z_3	3	-	0	0	1	0	0	1	-1	-1	$\{s_3\}$	1	-
Z_4	4	-	0	0	0	1	0	0	1	0	$\{s_4\}$	2	-
Z_5	5	-	0	0	0	0	1	0	1	0	$\{s_5\}$	2	-
Z_6	6	1	1	0	2	0	0	0	-2	0	$\{s_1,s_3\}$	2	$1Z_1+2Z_3$
Z_7	7		0	1	1	0	0	0	-1	0	$\{s_2,s_3\}$	2	$1Z_2+1Z_3$
Z_8	8	2	1	0	2	2	0	0	0	0	$\{s_1,s_3,s_4\}$	-	$2Z_4+1Z_6$
Z_9	9		0	1	1	1	0	0	0	0	$\{s_2,s_3,s_4\}$	-	$1Z_4+1Z_7$
Z_{10}	10		1	0	2	0	2	0	0	0	$\{s_1,s_3,s_5\}$	-	$2Z_5+1Z_6$
Z_{11}	11		0	1	1	0	1	0	0	0	$\{s_2,s_3,s_5\}$	-	$1Z_5+1Z_7$

Satz 8 zeigt auf, wie P-Varianten mit maximaler Netzüberdeckung zu ermitteln sind.

Satz 8

Eine P-Invariante mit einer maximalen Anzahl besetzter Plätze erhält man durch Bildung einer Linearkombination, die jede mit dem Algorithmus von Martinez ermittelte minimale Support P-Invariante mindestens einmal einbezieht.

Beweis

Seien $\phi_{P1} =(p_{11},p_{12},...,p_{1m})^T$, $\phi_{P2}=(p_{21},p_{22},...,p_{2m})^T$, ..., $\phi_{Pn}=(p_{n1},p_{n2},...,p_{nm})^T$ minimale Support P-Invarianten. Für die Linearkombination ϕ_{Pges}, die jede minimale Support P-Invariante mindestens einmal einbezieht, gilt dann:

$$\Theta_{Pges} = \sum_{i=1}^{n} \alpha_i \Theta_{Pi} = (\alpha_1 p_{11} + \alpha_2 p_{21} + ... + \alpha_n p_{n1}, ..., \alpha_1 p_{1m} + \alpha_2 p_{2m} + ... + \alpha_n p_{nm})^T$$

mit $p_{ab} \geq 0$ für $1 \leq a \leq n, 1 \leq b \leq m$

und $\alpha_i > 0$

Besetzt nun eine minimale Support ϕ_{Px}-Invariante den Platz y, d.h. $p_{xy}>0$, so bestimmt sich das y-te Element von ϕ_{Pges} zu $\alpha_1 p_{1y}+\alpha_2 p_{2y}+...+\alpha_x p_{xy}+...+\alpha_n p_{ny}$ (>0). Es existiert somit kein Platz, der von einer minimalen Support P-Invariante, nicht aber von ϕ_{Pges} besetzt wird. Satz 7 schließt P-Invarianten, die weitere Plätze umfassen, aus.

Da die Ergebnismenge des Algorithmus von Martinez auch P-Invarianten umfassen kann, die keine minimalen Support P-Invarianten sind, wird im Folgenden eine von Satz 8 leicht abgewandelte Formulierung gewählt, die eine algorithmisch sehr effiziente Umsetzung ermöglicht.

Satz 9

Eine P-Invariante mit einer maximalen Anzahl besetzter Plätze erhält man durch Bildung einer Linearkombination, die jede mit dem Algorithmus von Martinez ermittelte P-Invariante mindestens einmal einbezieht.

Beweis
Unter Einbeziehung nicht minimaler Support-Invarianten ist der Beweis analog zu dem von Satz 8.

Der Algorithmus von Martinez ist in der Referenzimplementierung des DevControl-Gateways umgesetzt (vgl. Kapitel 5.2). Zu Petrinetzmodellen beliebiger DevControl-Geräte wird die Ergebnismatrix in oben dargestellter Form ausgegeben. Die Angaben sind rein informativ. Sie können z. B. im Rahmen der Entwicklung von DevControl-Geräten herangezogen werden, um auf dieser Basis in der späteren Betriebsphase die kontinuierliche Überprüfung auf Markenkonstanz vorzunehmen. Die Entscheidung, ob und gegebenenfalls für welche Netzbereiche die Selbstüberwachung stattfindet, liegt im Ermessen des jeweiligen Geräteherstellers. Das DevControl-Gerät Jukebox ist beispielhaft als selbstüberwachend ausgeprägt. Diesbezüglich sei auf die weiterführenden Erläuterungen in Kapitel 6.6 verwiesen.

Kapitel 5

Referenzimplementierung

Zur Referenzimplementierung zählen mehrere Bestandteile, die hiermit verbundenen Programme sind durchgängig in C# formuliert. Für die nahtlose Integration der DevControl-Funktionalität in Browser-Umgebungen wurde ein Plugin für den Internet Explorer entwickelt. Hierauf wird in Kapitel 5.1 eingegangen. Dem Gateway kommt als Mittler zwischen steuernden und zu steuernden Geräten eine entscheidende Rolle zu. Der zweigestufte Ansatz – basierend auf DevXML und DevHTML – wird in der Referenzimplementierung voll unterstützt. Der Lösungsansatz mit den zugrunde liegenden Technologien stellt eine Möglichkeit der DevControl-Konzeptumsetzung dar, die eingehend diskutiert wird. Den Abschluss des fünften Kapitels bilden Ausführungen über die Einbindung von Mediastreams.

5.1 Browser-Plugin

Gemäß Architekturkonzept wird die generische Gerätesteuerung von einem Browser aus vorgenommen. Im ersten Schritt ist das im Netzwerk befindliche DevControl-Gateway zu kontaktieren und wahlweise der DevXML- oder DevHTML-Einstiegspunkt zu verwenden. Nähere Informationen hierzu sind in Kapitel 5.2 vorzufinden. Es ist naheliegend, die manuelle Adresseingabe durch einen Automatismus abzulösen.

Machbarkeit und Praktikabilität werden anhand eines für den Internet Explorer entwickelten Plugins nachgewiesen. Konzeptuell ist die Übertragbarkeit auf andere Browser gegeben. Für die .NET-basierte Entwicklung von IE-Plugins steht das Framework „Simple Plug-In Creator for Internet Explorer“ (SpicIE) zur Verfügung. Das DevControl-Plugin zeichnet sich durch eine vertikal angeordnete Toolbar aus, die alle für den Zugriff auf das DevControl-Gateway benötigten Funktionen bietet. Hierzu zählen die Suche nach dem DevControl-Gateway durch Rescan des lokalen Netzes und der Aufruf des DevXML- sowie des DevHTML-Control Points.

Ein vom Anwender initiierter Rescan des lokalen Netzes ist nur bei nicht ordnungsgemäßem Betrieb des DevControl-Gateways erforderlich. Dies ist zum Beispiel der Fall, wenn der als DevControl-Gateway fungierende Rechner abrupt ausgeschaltet wird. Im Regelbetrieb wird vollautomatisch ermittelt, ob ein DevControl-Gateway verfügbar ist. Sofern ein DevControl-Gateway detektiert wurde, können die zugehörigen Control Points aufgerufen werden. Zur Ermittlung der DevXML- und DevHTML-Einstiegspunkte greift das Plugin auf die vom DevControl-Gateway implementierten UPnP-Funktionen *GetDevXMLUrl()* und *GetDevHTMLUrl()* zu. Vergleiche hierzu die Ausführungen in Kapitel 4.2.2.

Das Plugin verwendet den Intel UPnP-Stack und liegt als Dynamic Link Library vor. Es lässt sich mit der Anweisung

```
regasm DevControlPlugin.dll /codebase
```

registrieren. Mit der Anweisung

```
regasm DevControlPlugin.dll /unregister
```

wird die Registrierung aufgehoben. Da im Rahmen der vorliegenden Arbeit die konzeptuellen Aspekte im Vordergrund stehen, wird von der Integration obiger Verwaltungsfunktionen in eine noch komfortabler zu handhabende Installationsroutine abgesehen.

5.2 Gateway

Mit der Anwendung *ControlPoint* wird das Kernstück des Architekturkonzeptes, das DevControl-Gateway, realisiert. Für die Steuerung unter Verwendung der Spezifikationen DevHTML und DevXML sind zwei Einstiegsmöglichkeiten vorhanden. Nach Ausführung der Server-Applikation auf einem Netzwerkrechner mit der Netzwerkkennung *CPServer* kann die HTML-Benutzerschnittstelle mit allen gängigen Browsern über den Link *http://CPServer:8181/cp/ShowDevices.html* aufgerufen werden. Das DevXML-Interface wird über *http://CPServer:8181/parser/Navigator.html* bereitgestellt.

Der Navigationsbereich des DevHTML-Control Point (siehe Abbildung 69) befindet sich links. Im oberen Auswahlfenster kann zwischen dem Aufruf einer Informationsseite, der Hauptseite der TU Freiberg sowie dem Rescan des DevControl-Netzwerkes gewählt werden. Für die Anzeige der hiermit verbundenen Inhalte wird der rechte Fensterbereich genutzt. Das untere Auswahlfenster stellt die im Netzwerk befindlichen UPnP-Geräte mit ihren eingebetteten Geräten, Diensten und Web-Interface-Methoden – gekennzeichnet durch das Präfix WI_ – in der Baumansicht dar. Hierbei werden nur solche Dienste aufgeführt, die mindestens eine Web-Interface-Methode implementieren. Die entsprechenden Knoten sind durch Farbverlauf von grün nach blau symbolisiert. Diese Farbgebung gilt auch für das jeweils unmittelbar übergeordnete Gerät. Alle anderen Knoten sind in Grautönen gehalten. Hierunter fallen UPnP-Geräte außerhalb der DevHTML-Spezifikation. Die Selektion einer WI_-Methode bewirkt deren Aufruf in einem separaten Browser-Fenster. Geräte mit einer Presentation Page werden durch unterstrichene Gerätebezeichnungen hervorgehoben. Nach Einfachklick hierauf wird die Presentation Page im rechten Fensterbereich angezeigt.

Die korrekte Darstellung der vom DevControl-Gateway erzeugten HTML-Seite bedingt die browserseitige Aktivierung von Javascript. Nur so kann die dynamisch generierte Baumstruktur zur Navigation eingesetzt werden[144]. Mediastreams gemäß der Spezifikation in Kapitel 4.2.3 verfügen über WI_-Methoden. Damit zählen sie zur Gruppe der DevHTML-Geräte – in der Folge kann über das oben beschriebene Auswahlfenster ihre Funktionalität genutzt werden. Sofern DevHTML-Geräte die Methode *string GetAdditionalName()* innerhalb des Dienstes *urn:tu-freiberg.de:service:general:1* implementieren, kann diese zur näheren Namensqualifizierung vom DevControl-Gateway verwendet werden.

[144] Die Realisierung der Baumstruktur erfolgt unter Rückgriff auf die JavaScript-Bibliothek Sothink Tree Menu 2.9 der Firma SourceTec Software Co.

Diese Zusatzinformation, die z. B. bei Mediastreams vorhanden ist, ergänzt dann den Standard-UPnP-Gerätenamen innerhalb des DevHTML-Control Point. Sie ist in Klammern gesetzt und dient der einfacheren Zuordenbarkeit. Im Ergebnis führt dies zu aussagekräftigen Bezeichnungen wie „Webcam (Office, 1st floor)“.

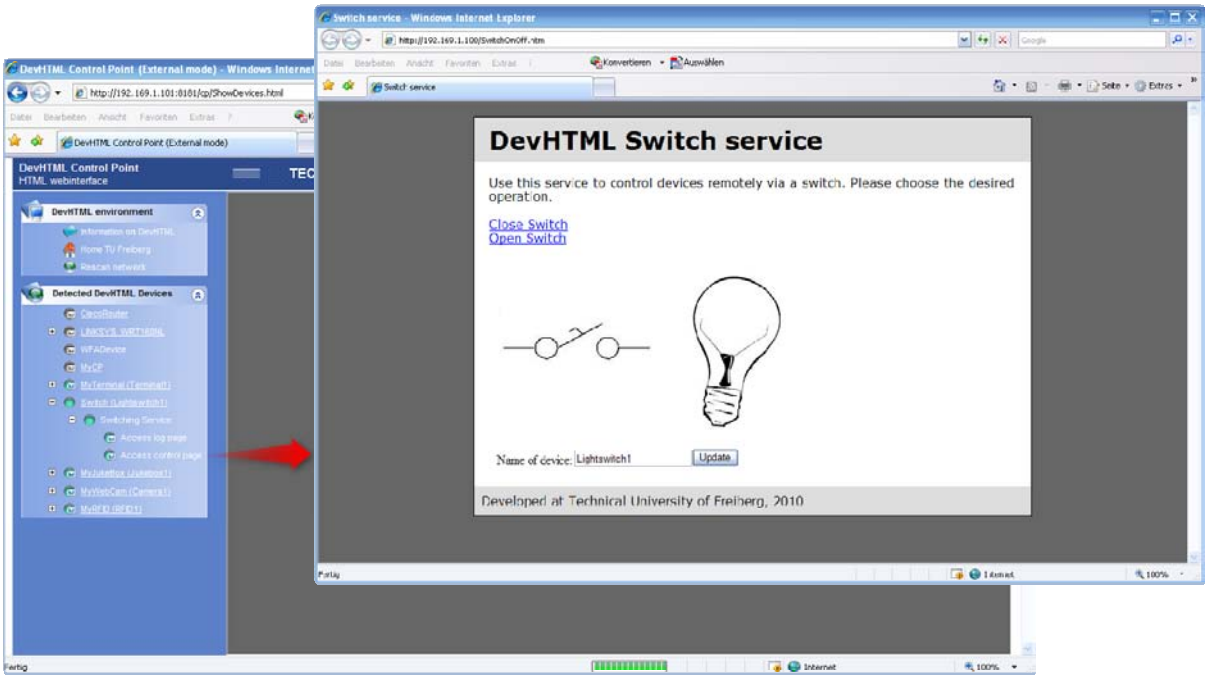


Abbildung 69 DevHTML-Control Point zur Steuerung von Geräten mit Web Interfaces. Nach Auswahl von Gerät und Dienst in der Baumstruktur ist das gewünschte Web Interface zu selektieren. Dieses wird in einem separaten Browserfenster angezeigt. Im Beispiel ist das zum Gerät „Switch“ und zum Dienst „Switching Service“ gehörige Web Interface „Access control page“ zu sehen.

Geräte für die Autoidentifikation gemäß der Spezifikation in Kapitel 4.2.4 fügen sich ebenfalls in den DevHTML-Control Point ein. Die Anwendung *RFIDReader* mitsamt zugehöriger Hardware demonstriert die Funktionsweise. Nach deren Start greift das DevControl-Gateway im 500 ms-Takt auf die Methode *string GetRFIDData()* des Dienstes *urn:tu-freiberg:service:autoid:1* zu. Befindet sich ein Tag mit einem gültigen UDN in der Reichweite des RFID-Lesegerätes, so wird dieser UDN als Ergebnis zurückgeliefert, andernfalls ein Leerstring. Im ersten Fall führt der DevHTML-Control Point im Anschluss einen Abgleich mit den UDNs der von ihm verwalteten Geräte durch. Bei Übereinstimmung wird das entsprechende Gerät automatisch selektiert. Die Formaterkennung der Safari-Browser-Engine kann beim Test von Zeichenketten auf Gleichheit zu Komplikationen führen. Um die mögliche Reformatierung von UDNs innerhalb des vom DevHTML-Control Point bereitgestellten HTML-Codes auszuschließen, ist folgendes <*meta*>-Tag aufzunehmen[145]:

```
<meta name="format-detection" content="telephone=no">
```

[145] Wagner: Safari and WebKit Development for iPhone OS 3.0, S. 245.

Die RFID-Referenzimplementierung lässt darüber hinaus die Aktivierung und Deaktivierung des Erkennungsmodus mittels Web-Interface-Methode zu. Hierfür ist die Auswahl „Set current mode“ innerhalb des Dienstes „Configuration“ zu treffen. In Kapitel 5.4 wird näher auf die hard- und softwaretechnische Realisierung eingegangen.

Der DevXML-Control Point, der eine weitere Einstiegsmöglichkeit bietet, ist in Silverlight realisiert. Infolgedessen wird auf dem vom Anwender genutzten Endgerät ein entsprechendes Browser-Plugin benötigt. Nach Aufruf von *http://CPServer:8181/parser/Navigator.html* ist zwischen zwei Bedienoberflächen zu wählen. Die „Engineering View“ hat den Zweck, interne Abläufe im Zusammenhang mit der Auswertung von DevXML-Code zu verdeutlichen. Insgesamt stehen drei von links nach rechts positionierte Comboboxes mit jeweils einem unterhalb angeordneten Anzeigebereich zur Verfügung. Nachdem mit der ersten Combobox ein DevXML-Gerät selektiert wird, erscheint das zugehörige Petrinetz im zugeordneten Anzeigebereich. Die hier belegten Plätze, die gerade eingenommene Teilzustände des Gerätes reflektieren, stehen wiederum in der zweiten Combobox zur Auswahl. Hier führt die Selektion eines Platzes zur Anzeige der ihn charakterisierenden Statusvariablen sowie Medienströme. Feuerbare Transitionen, die aus der aktuellen Petrinetzbelegung resultieren, sind in der dritten Combobox auswählbar.

Im dazugehörigen Anzeigebereich besteht die Möglichkeit, mit dem Anwender unter Nutzung der in Kapitel 4.3.2.3 „Elemente zur Transitionsspezifikation“ beschriebenen Steuerelemente zu interagieren. Auf diese Weise können Zustandsänderungen herbeigeführt werden. In der „Engineering View“ sind Auswirkungen auf Platzmarkierungen sofort ersichtlich. Durch Setzen des Hakens im Optionsfeld „Load optional UI code if available“ wird festgelegt, dass eventuell vorhandener externer UI-Code heranzuziehen ist. Andernfalls wird das UI mit all seinen Elementen vollständig autogeneriert. Abbildung 70 zeigt das UI des DevXML-Control Point.

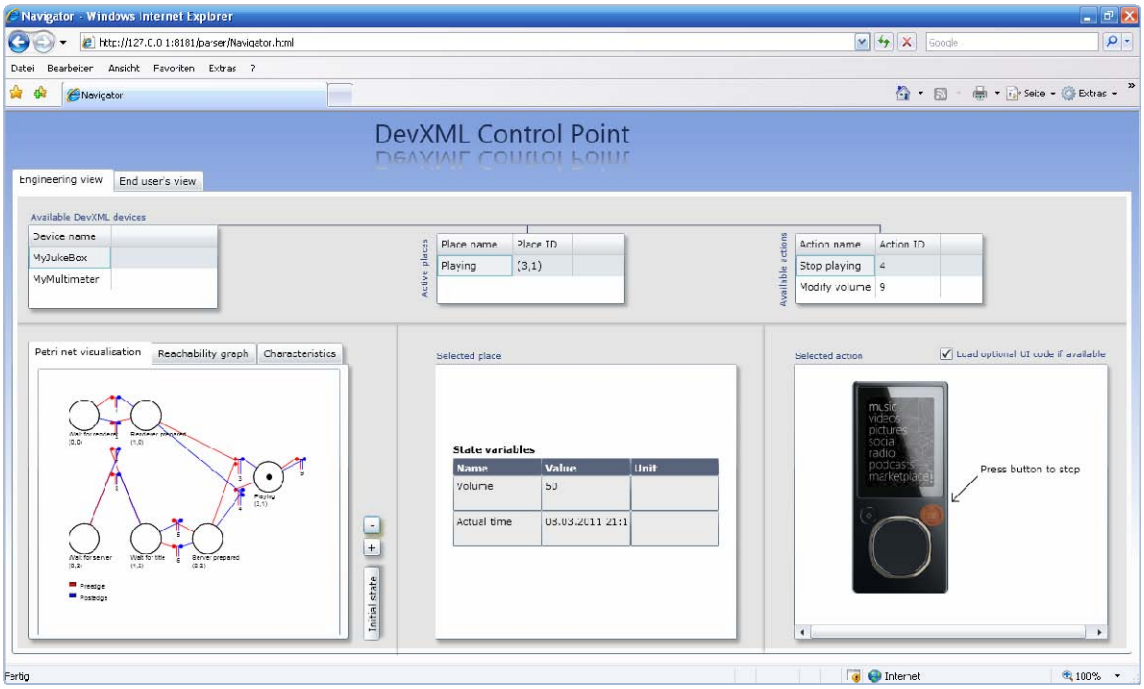


Abbildung 70 DevXML-Control Point in der „Engineering View“. Gerät, Platz und Transition wurden ausgewählt. Die korrespondierenden Anzeigebereiche für das Petrinetz, die platzbezogenen Statusvariablen und Medienströme sowie die transitionsbezogene Benutzerschnittstelle sind im unteren Bildschirmbereich angeordnet.

Neben der graphischen Repräsentation des Petrinetzes sind im links unten befindlichen Anzeigebereich durch Klick auf die entsprechenden Reiter auch Erreichbarkeitsgraph und die Ergebnismatrix des Algorithmus von Martinez darstellbar. Diesen kommt bei der Nutzerführung in der für den Produktiveinsatz bestimmten „End User's View" eine hohe Bedeutung zu.

In der „End User's View" wählt der Anwender die gewünschte Zieltransition. Aufgeführt sind nur solche Transitionen, die Interaktionen über Steuerelemente zulassen – d. h. über das Subelement <*transitioncontrol*> verfügen. Des Weiteren muss das zum DevXML-Element <*transition*> gehörige Attribut *targetTransition* den Wahrheitswert *true*, der gemäß Voreinstellung gilt, annehmen. Anschließend wird geprüft, aus welchen Netzmarkierungen diese feuerbar ist. Grundlage für die Pfadermittlung im Rahmen der Nutzerführung ist die mit minimaler Strecke erreichbare Markierung. Gegeben sei beispielsweise das Petrinetz aus Abbildung 71 mit t_3 als Zieltransition. t_3 ist aus den Markierungen $(0\ 0\ 1\ 1\ 0)^T$ sowie $(0\ 0\ 1\ 0\ 1)^T$ feuerbar. Vom aktuellen Zustand $(1\ 0\ 0\ 1\ 0)^T$ hat $(0\ 0\ 1\ 1\ 0)^T$ die kürzere Entfernung. Vereinbarungsgemäß wird somit hier hingeführt.

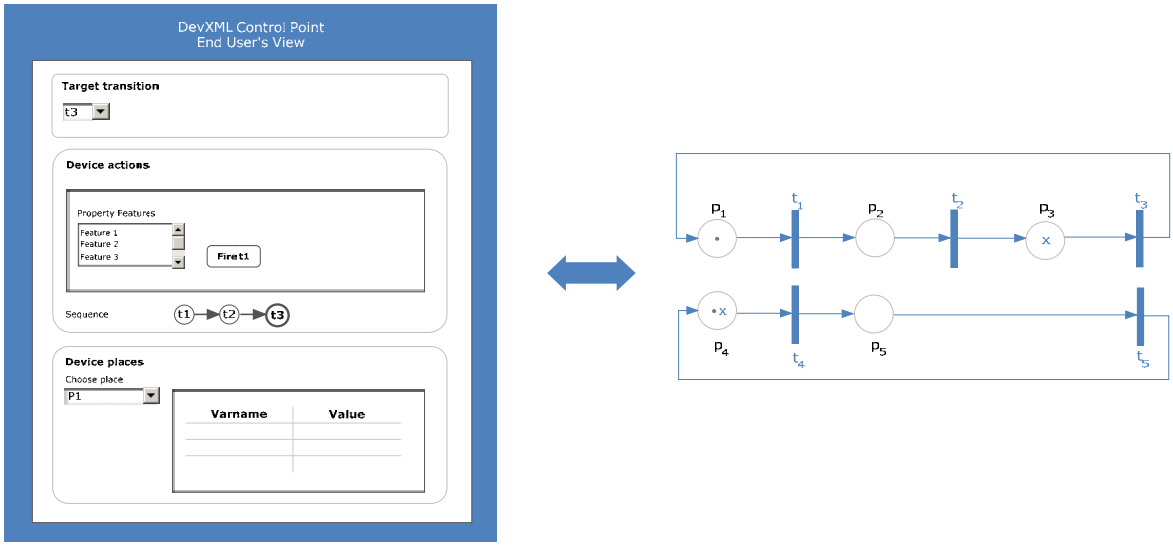


Abbildung 71 Schematische Darstellung der „End User's View" des DevXML-Control Point mit zugehörigem Petrinetz. Ausgehend vom aktuellen Zustand $(1\ 0\ 0\ 1\ 0)^T$ gibt der Benutzer die Zieltransition t_3 vor. Die erforderliche Transitionsfolge t_1, t_2, t_3 – gekennzeichnet durch minimale Wegstrecke – wird automatisiert ermittelt. Sie dient im Weiteren der Nutzerführung.

Die markierten Plätze der Ausgangsmarkierung sind durch das Zeichen „·", die von $(0\ 0\ 1\ 1\ 0)^T$ durch das Zeichen „x" gekennzeichnet. In der linksseitig angeordneten Benutzerschnittstelle, die schematisch dargestellt ist, symbolisieren Kreise die zu feuernden Transitionen t_1, t_2 und schließlich – vorgabegemäß – t_3.

Generell wird durch das DevXML-Gateway jeder Zustandsübergang mit dem ihn bewirkenden Ereignis unter Angabe des aktuellen Zeitstempels in einer Datei geloggt. Dies erleichtert die Ursachenermittlung bei möglicherweise nicht ordnungsgemäßem Verhalten von zu steuernden Geräten.

5.3 Mediastreams

Mit der Anwendung *Webcam* liegt eine Referenzimplementierung für Mediastreams vor. Diese lässt sich in Verbindung mit handelsüblichen USB-Kameras einsetzen. Die Funktionalität wird über das vom UPnP Basis-Device abgeleitete Gerät *Webcam*, das die Dienste *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:mediastream:1* beinhaltet, zur Verfügung gestellt. Das zugehörige Klassendiagramm zeigt Abbildung 72. Alle erforderlichen Aktionen gemäß Kapitel 4.2.3 „Interface von Mediastreams“ sind implementiert. Die Angabe für „Name of Webcam“ im Hauptfenster der Anwendung bestimmt den Rückgabewert der Aktion *string GetAdditionalName()* innerhalb von *urn:tu-freiberg.de:service:general:1*. Der Eintrag ist mit der Zeichenkette „Camera 1“ vorbelegt.

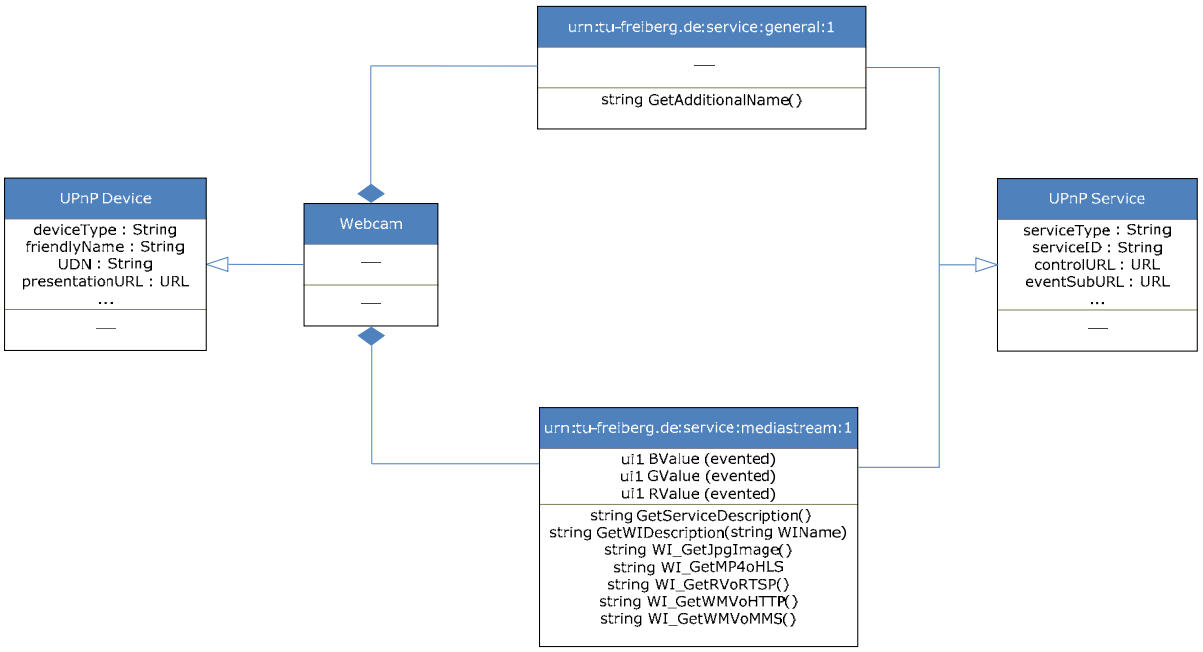


Abbildung 72 Klassendiagramm des DevControl-Gerätes Webcam. Neben dem allgemeinen Dienst *urn:tu-freiberg.de:service:general:1* gemäß Kapitel 4.2.1 umfasst Webcam den gerätespezifischen Dienst *urn:tu-freiberg.de:service:mediastream:1*.

Im Dienst *urn:tu-freiberg.de:service:mediastream:1* sind alle Aktionen und Statusvariablen mit Bezug zum Streaming zusammengefasst. Die Aktion *string GetServiceDescription()* liefert den Rückgabewert „Streaming service“. Der Rückgabewert der Aktion *string GetWIDescription(string WIName)* variiert in Abhängigkeit des Parameters *WIName*. Den Zusammenhang zeigt Tabelle 48 auf.

Tabelle 48 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetJpgImage	Jpg sequence
WI_GetMP4oHLS	MP4 over HTTP Live stream
WI_GetRVoHTTP	RV over RTSP
WI_GetWMVoHTTP	WMV over HTTP
WI_GetWMVoMMS	WMV over MMS
sonst	Unknown WI

Sofern innerhalb des zur Anwendung *Webcam* gehörigen Gruppenrahmens „JpgStreaming“ eine Kamera ausgewählt wurde, liefert die Aktion *string WI_GetJpgImage()* eine nicht-leere URL zurück. Die referenzierte HTML-Seite ist JavaScript-basiert und bietet eine Sequenz von Bildern im Jpg-Format an. Der Zugriff auf die Kamera erfolgt mittels DirectShow. Basiselemente der DirectShow-Architektur sind Filter, die sich in Quell-, Transformations- und Anzeigefilter untergliedern, sowie die mit ihnen assoziierten Pins. Letztgenannte Objekte, die als Anknüpfungspunkt für Ein- oder Ausgaben fungieren, dienen der Beschreibung von Medienströmen zwischen den Filtern. Im Graph der Anwendung *WebCam* ist der Quellfilter von der Kategorie „Video Capture Sources“.

Um die Aufteilung des Datenstromes zu bewirken, wird dieser ausgehend vom „Capture“-Pin dem Eingangspin des Filters „Smart Tee“ zugeleitet. Die Speisung des „Grabber“-Filters über den „Preview“-Pin ermöglicht den Zugriff auf das aktuell erfasste Bild, das für die Berechnung des durchschnittlichen Farbwertes herangezogen wird[146]. Die mit dem Zusatz *evented* deklarierten Statusvariablen *RValue*, *GValue* und *BValue* geben den Rot-, Grün- und Blau-Farbanteil wieder. Hierfür abonnierte Kontrollpunkte erhalten bei Änderungen Ereignismeldungen.

Der Filter „Video Renderer“ ist Grundlage für die Anzeige des von der Webcam generierten Livestreams. Da Videohardware typischerweise RGB32-Farbwerte verarbeitet und Webcams in der Regel RGB24-Farbwerte bereitstellen, befindet sich im Beispiel zwischen dem „Smart Tee“ und dem „Video Renderer“ ein zusätzlicher Filter zur Farbraumkonvertierung. Dieser ist mit „Color Space Converter“ bezeichnet. Der vom „Smart Tee“ führende „Capture“-Pin beinhaltet im Gegensatz zum „Preview“-Pin Zeitstempel. Dies ist z. B. bei der Speicherung von Mediendaten nützlich, um auf diese Weise trotz möglicher Latenzen infolge des „SmartTee“-Filters Zeittreue sicherzustellen. Für die Anzeige des Livestreams im Vorschaubereich werden die übermittelten Zeitstempel hingegen nicht benötigt[147]. Auch für Grabbing sind Zeitstempel überflüssig. „Capture“- und „Preview“-Pin sind daher vertauschbar. Die beschriebenen Zusammenhänge sind in Abbildung 73 dargestellt.

[146] Linetsky: Programming Microsoft DirectShow, S. 83 ff.

[147] Erreichen Frames mit Zeitstempel den „Video Renderer“, werden diese bei hinreichend großen Latenzen empfängerseitig sogar verworfen. Dieser Effekt tritt bei Verwendung des „Preview“-Pins nicht auf.

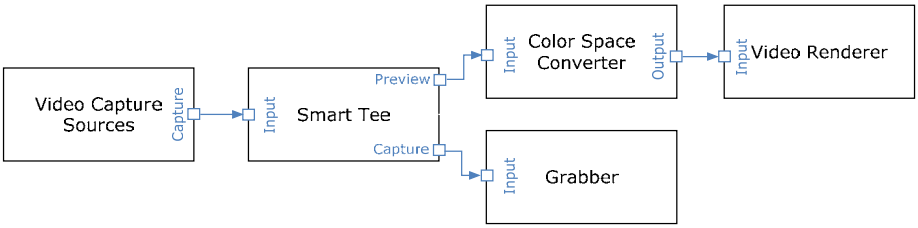


Abbildung 73 DirectShow-Graph zur Realisierung des Zugriffs auf Livestreams mit Grabbing. Das Signal der Video Capture Source – im Beispiel eine Webcam – wird vom Smart Tee-Filter aufgeteilt. Zum einen nimmt der Color Space Converter die Farbraumkonvertierung vor, sodass der Datenstrom vom Video Renderer angezeigt werden kann. Zum anderen liefert der Smart Tee-Filter den Input für den Grabber-Filter. Dieser ermöglicht die Weiterverarbeitung des aktuell erfassten Bildes.

Die vier weiteren Streaming-Varianten MP4 über HTTP Live Streaming, RV über RTSP, WMV über HTTP und WMV über MMS werden von der Anwendung Webcam mittelbar unterstützt. Innerhalb des Gruppenrahmens „Advanced Streaming“ können entsprechende Verweise gesetzt werden. Die Medieninhalte sind dann von externen Programmen wie Windows Media Encoder oder Helix Universal Server v14 bereitzustellen. Auf die hier gemachten Angaben greifen die Aktionen *string WI_GetMP4oHLS()*, *string WI_GetRVoRTSP()*, *string WI_GetWMVoHTTP()* sowie *string WI_GetWMVoMMS()* zurück. Gemäß Kapitel 4.2.3 muss mindestens eine der insgesamt fünf Kombinationen aus Kodierungsformat und Übertragungsprotokoll umgesetzt sein. Wie bereits in Kapitel 2.4.3 ausgeführt, kann die Anordnung der einzelnen Komponenten so erfolgen, dass multivalentes Streaming möglich ist (siehe Abbildung 46). Das Multiplexen des Datenstroms einer Kamera ist Voraussetzung für den zeitgleichen Zugriff durch mehrere Anwendungen[148].

5.4 Autoidentifikation

Im Rahmen der Referenzimplementierung stützt sich die Autoidentifikation auf RFID. Die gleichnamige Anwendung greift unter Verwendung der vom Hersteller MEGASET mitgelieferten Bibliotheken auf das Lese- und Schreibgerät ICR40C/RW zu. Die Hardware wird hierbei über eine RS-232-Schnittstelle angebunden. Der genutzte Port ist in der Anwendung RFID anzugeben. Unterstützt wird die Transponder-Serie Titan.

Voraussetzung für die Autoidentifikation von Geräten ist die Bereitstellung des jeweiligen UDN auf einem Tag. Der Schreibvorgang kann im Gruppenrahmen „Data writing“ durch Auswahl eines detektierten Gerätes und anschließender Betätigung des Buttons „Write UDN to tag“ initiiert werden. Das Interface für die Autoidentifikation gemäß Kapitel 4.2.4 wird mit der Aktion *string GetData()* innerhalb des Dienstes *urn:tu-freiberg.de:service:autoid:1* unterstützt. Siehe hierzu Abbildung 74.

[148] Es eignet sich beispielsweise das Programm „broadcaster StudioPRO“

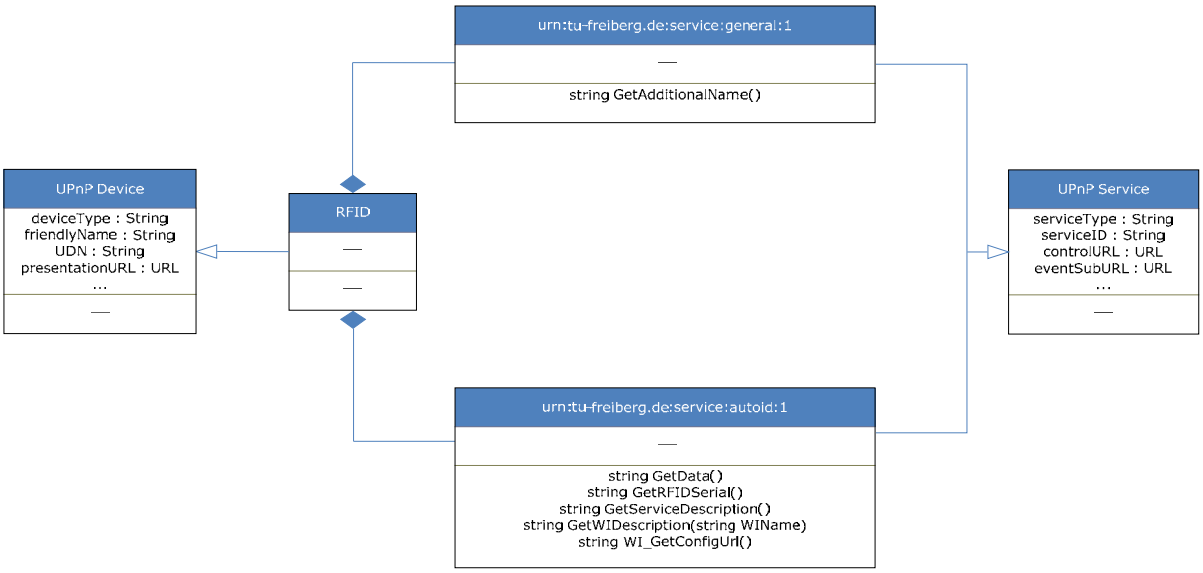


Abbildung 74 Klassendiagramm des DevControl-Gerätes RFID. Neben dem allgemeinen Dienst *urn:tu-freiberg.de:service:general:1* gemäß Kapitel 4.2.1 umfasst RFID den gerätespezifischen Dienst *urn:tu-freiberg.de:service:autoid:1*.

Alle weiteren Aktionen gehen über die Spezifikation des Interface für die Autoidentifikation hinaus. Mit *string GetRFIDSerial()* kann die Seriennummer eines Tags ermittelt werden. Bei der Titan-Serie handelt es sich um einen 32-Bit Fixcode. Die übrige Funktionalität dient der näheren Beschreibung des Dienstes sowie seiner Webinterfaces. Der Rückgabewert der Aktion *string GetServiceDescription()* lautet „Configuration“. Die Aktion *string GetWIDescription(string WIName)* liefert die in Tabelle 49 genannten Rückgabewerte.

Tabelle 49 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetConfigUrl	Set current mode
sonst	Unknown WI

Die im Gruppenrahmen „Data Reading“ befindlichen Buttons „Read tag data“ sowie „Read tag serial number“ sind dazu bestimmt, innerhalb der Anwendung die auf dem Tag gespeicherten Daten bzw. seine Seriennummer auszulesen. Auf diese Weise können die Rückgabewerte der Aktionen *string GetData()* und *string GetRFIDSerial()* unmittelbar überprüft und beispielsweise Schreibfehler aufgedeckt werden. Wie in Kapitel 5.2 „Gateway“ geschildert, greift ein DevHTML-Control Point in regelmäßigen Abständen auf alle im Netzwerk vorhandenen Geräte für die Autoidentifikation zu. Die RFID-Anwendung bietet die Möglichkeit, den UDN-Abgleich zu unterbinden. Hierzu ist vom Control Point aus die Aktion „Set curent mode“ innerhalb des Dienstes „Configuration“ zu wählen. Ein Klick auf „Deactivate reader“ in der zugehörigen HTML-Seite führt dazu, dass der Zugriff auf das RFID-Lesegerät unterbleibt und die Aktion *string GetData()* einen Leerstring zurückgibt.

Kapitel 6

Experimentelle Validierung anhand ausgewählter DevControl-Geräte

Anhand ausgewählter DevControl-Geräte wird das entwickelte Framework experimentell validiert. Zahlreiche Beispiele demonstrieren die Vielseitigkeit von DevHTML und DevXML. Der zweigliedrige Ansatz wird mit Ausnahme der rein DevHTML-basierten Anwendung Terminal von allen Implementierungen unterstützt. Die Terminal-Anwendung ermöglicht die mittelbare Steuerung beliebiger Geräte über IP-Netze. Voraussetzung ist der Gerätezugriff von Windows. Für die Aufnahme derartiger Geräte in eine DevControl-Umgebung bedarf es keiner Anpassungen. Die skizzierten Vorteile von DevHTML und DevXML – wie die automatisierte Geräteidentifikation, einheitliche User Interfaces etc. – kommen in diesem Fall jedoch nicht zum Tragen. Es existieren zwei weitere Integrationspfade. Zum einen ist das die Neuentwicklung von DevControl-Geräten, zum anderen die Erweiterung bereits bestehender Geräte um DevControl-Funktionalität. Somit ergeben sich insgesamt drei Use Cases, die in Abbildung 75 zusammengefasst sind. Mit der Terminal-Anwendung steuerbare sowie realisierte Geräte sind farblich hervorgehoben.

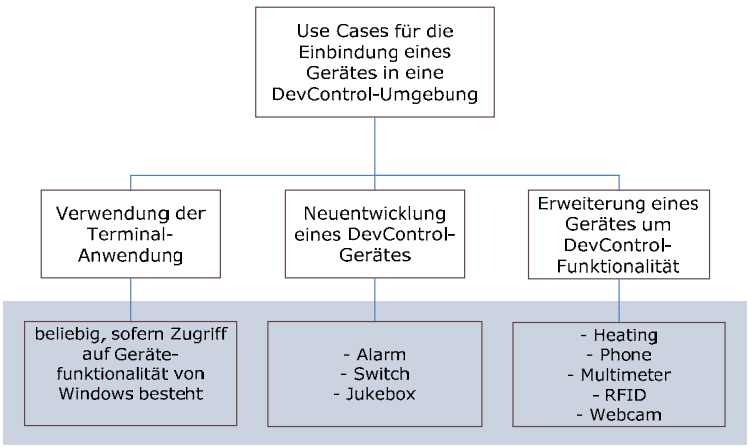


Abbildung 75 Ein Gerät kann auf drei Wegen in eine DevControl-Umgebung eingebracht werden. Anhand zahlreicher Entwicklungen werden die unterschiedlichen Ansätze vorgestellt.

Die einzelnen Schritte zur Erreichung von DevControl-Konformität werden für unterschiedliche Systeme erläutert. Es variieren die den Implementierungen zugrunde liegenden Programmiersprachen sowie die Geräteklassen. Die typischen Paarungen Standalone Mikrocontroller und Dynamic C, Mobiles Endgerät und Java sowie Windows-Anwendung und C# finden Berücksichtigung. Für die realisierten Geräte sind die diesbezüglichen Angaben in nachfolgender Tabelle aufgeführt.

Tabelle 50 Paarungen von Programmiersprachen und Geräteklassen der realisierten DevControl-Geräte. Die hiervon abgeleiteten Virtual Devices sind nicht aufgeführt.

DevControl-Gerät	Verwendete Programmiersprache	Geräteklasse
Terminal	C#	Windows-Anwendung
Alarm	C#	Windows-Anwendung
Phone	Java	Mobilgerät
Heating	C#	Windows-Anwendung
Multimeter	C#	Windows-Anwendung
Switch	Dynamic C	Standalone Mikrocontroller
Webcam	C#	Windows-Anwendung
RFID	C#	Windows-Anwendung
Jukebox	C#	Windows-Anwendung

Es ist zu beachten, dass den hier vorgestellten Anwendungen – mit Ausnahme von Terminal – eine Mittlerfunktion zukommt und die eigentliche Funktionalität durch zusätzliche Hardware bereitgestellt wird. Eine Windows-Anwendung kann auf einem Standard-PC, der nicht exklusiv mit der hierzu gehörigen Hardware verbunden ist, ausgeführt werden. Hieraus resultiert der Umstand, dass der Anwender im Vorfeld die jeweilige Geräteeinheit formieren und im Anschluss die Anwendung manuell starten muss. Bei kommerziellen Produkten sollte darauf hingewirkt werden, dass die Geräteeinheit bereits besteht. Dies ist z. B. mittels Kleinrechner zu bewerkstelligen. Somit wird im Produktivbetrieb der Forderung nach Plug and Play nachgekommen. Siehe Abbildung 76.

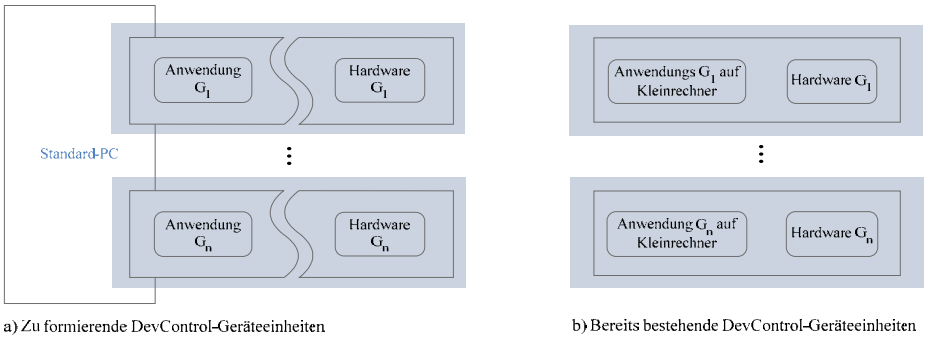


Abbildung 76 Zwischen einer Windows-Anwendung und der zugehörigen Hardware kann eine Nicht-Exklusivitätsbeziehung (Fall a) oder eine Exklusivitätsbeziehung (Fall b) bestehen. Bei kommerziellen Produkten ist auf Fall b hinzuwirken, um so den Einrichtungsaufwand für Endanwender zu minimieren.

Der Zugriff auf DevControl-Geräte ist lokal und über das Internet möglich. Grundlage hierfür ist die Portmapping-Fähigkeit des DevControl-Gateways. Nach Auswahl von „Portmapping“ im Menü „Options“ können die für die Steuerung eines DevControl-Gerätes relevanten internen Ports automatisiert auf externe abgebildet werden. Dies sind solche, die im Zusammenhang mit der Presentation Page sowie den WI_-Aktionen stehen. Darüber hinaus wird der vom DevControl-Gateway genutzte Port 8181 in analoger Weise gemappt, um den Abruf der DevHTML- und DevXML-Einstiegsseiten über das Internet zu gewährleisten. Abbildung 77 zeigt eine typische Anordnung, bei der zusätzlich ein DynDns-Server die aktuell gültige öffentliche IP-Adresse des DevControl-Gateways hinterlegt und über einen frei wählbaren Hostnamen verfügbar macht.

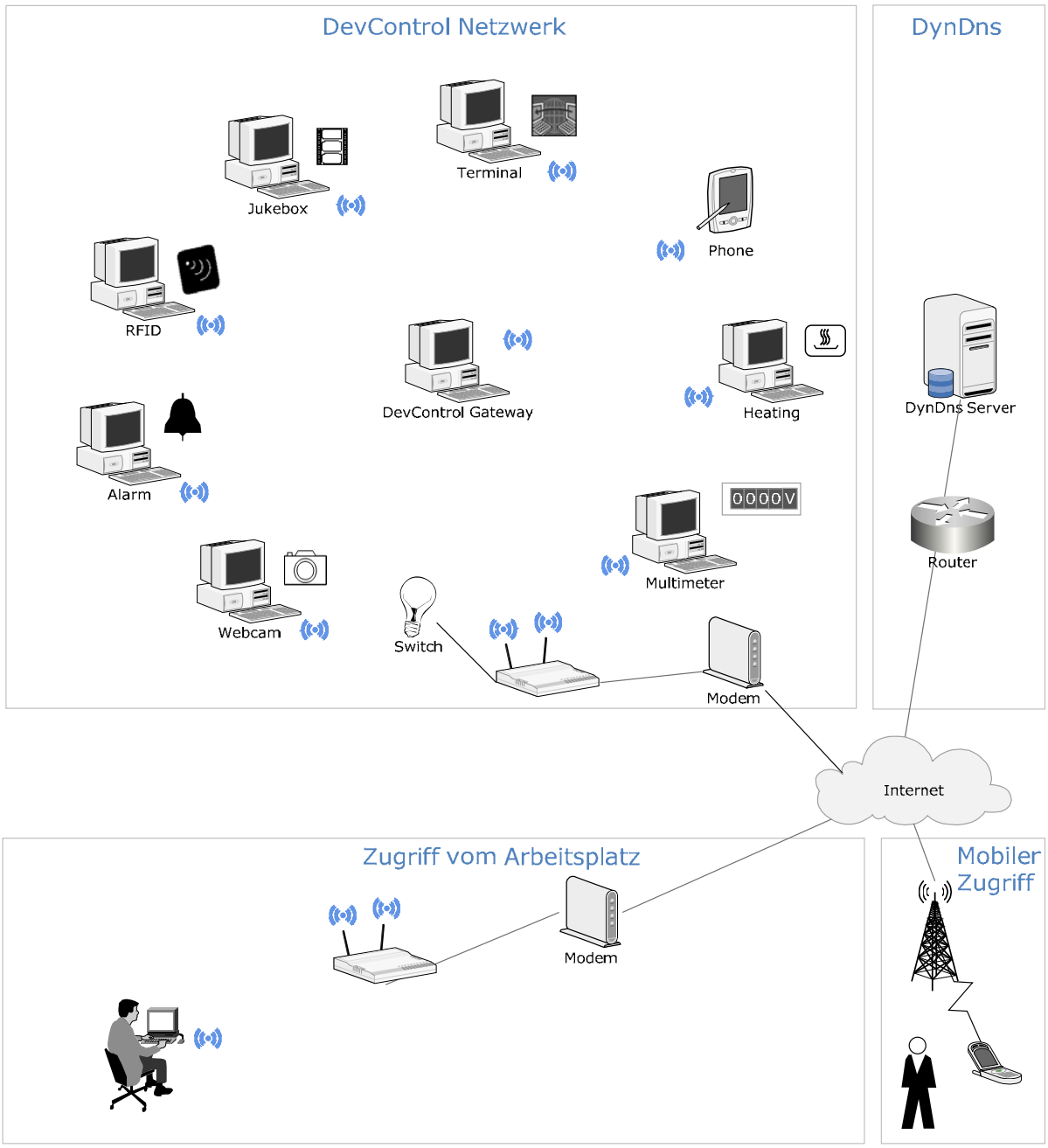


Abbildung 77 Mit der abgebildeten Anordnung ist der Internet-basierte Zugriff auf DevControl-Geräte unter Verwendung von Hostnamen möglich. Alle für die Steuerung eines DevControl-Gerätes relevanten Ports können hierbei vom DevControl-Gateway automatisiert gemappt werden.

Virtual Devices, die auf Teilfunktionalitäten obiger DevControl-Geräte zurückgreifen, sind Haptic Phone und Automated Switch. Haptic Phone bezieht RFID und Phone mit ein, um Tag-gesteuert den Wählvorgang zu initiieren. Webcam und Alarm bilden die Basis für Automated Switch. Schaltvorgänge können so in Abhängigkeit von Zeit und Helligkeit bewirkt werden. Die Realisierung der einzelnen DevControl-Geräte wird im Weiteren eingehend erläutert.

6.1 Switch

Das vorgestellte Architekturkonzept setzt auf TCP/IP-basierter Kommunikation auf. Daher bieten sich Embedded-Systeme an, die bereits über einen TCP/IP-Protokollstack verfügen. Zu nennen sind beispielsweise AVR-Mikrocontroller der Firma Atmel in Kombination mit dem TCP/IP-Mikrochip W3100A der Firma Wiznet[149] sowie Mikrocontroller der Firma Rabbit Semiconductor. Anhand des Entwicklungskits RCM4000 mit dem Modul RCM4010 der Firma Rabbit Semiconductor wird exemplarisch beschrieben, wie DevXML-Konformität erreicht werden kann. Das RCM4010-Modul weist eine Taktfrequenz von ca. 60 MHz auf und verfügt über 512 kByte SRAM sowie 512 kByte Flash-Speicher. Die Entwicklung von Programmen hierfür ist in Dynamic C vorzunehmen. Die mitgelieferten Bibliotheken beinhalten den UPnP-Stack, der Voraussetzung für DevControl ist, nicht. Daher umfasst die Eigenentwicklung ebenfalls die für den UPnP-basierten Kommunikationsprozess benötigte Funktionalität.

Das Modul wird mit der Spannung U_{ext} vom Prototyping Board versorgt. Durch Setzen der Ausgangsspannung auf 0 V oder U_{Be} an wählbaren Output-Pins lässt sich leicht ein Schalter realisieren. Da die Schaltströme und -spannungen des Moduls niedrig sind, werden diese ausschließlich für die Ansteuerung des Transistors $Trans_1$ verwendet. Ergänzend kommt eine Spannungsquelle U_{CE} zum Einsatz, die mit dem Relais Rel_1 verbunden ist. Auf diese Weise können Verbraucher größerer Leistung in einem weiteren Stromkreis geschaltet werden. Die gesamte Elektronik, die nicht zum Prototyping Board zählt, ist in einem Gehäuse, das über eine Steckdose für Verbraucher und einen Stecker für die Spannungsversorgung verfügt, untergebracht. Siehe hierzu Abbildung 78.

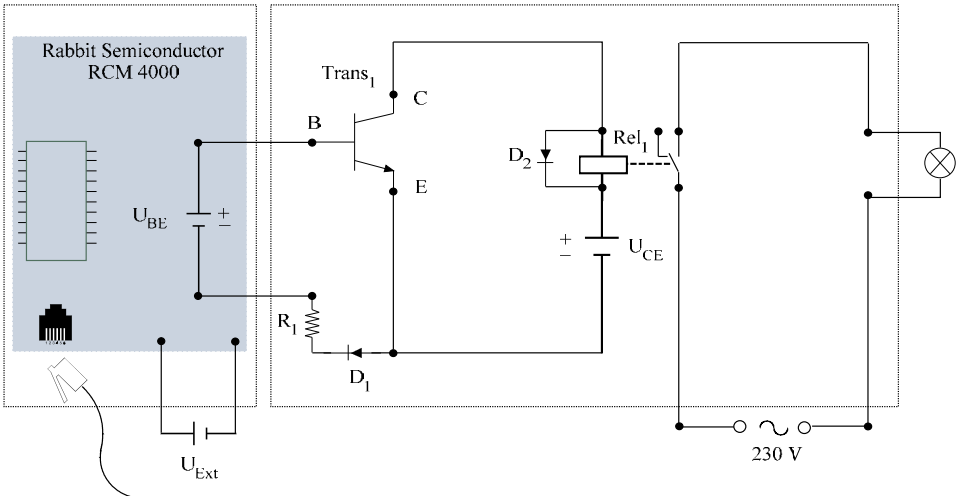


Abbildung 78 Prototyping Board RCM 4000 mit zusätzlicher Elektronik zum Schalten von Verbrauchern größerer Leistung. Der Transistor $Trans_1$ wird für die Ansteuerung des Relais Rel_1 verwendet.

Eine konkrete Ausführung zeigt Abbildung 79. Das Prototyping Board ist über ein zweiadriges Kabel direkt mit der in einer schwarzen Steckdose befindlichen Elektronik, die Transistor und Relais umfasst, verbunden. Für die Anzeige des aktuellen Gerätezustands sowie Schalthandlungen unter Verwendung von DevHTML eignet sich u. a. ein iPhone. Im Beispiel ist der Stromkreis geschlossen.

[149] Biller: Vernetzung mit dem Ethernet, S. 49 ff.

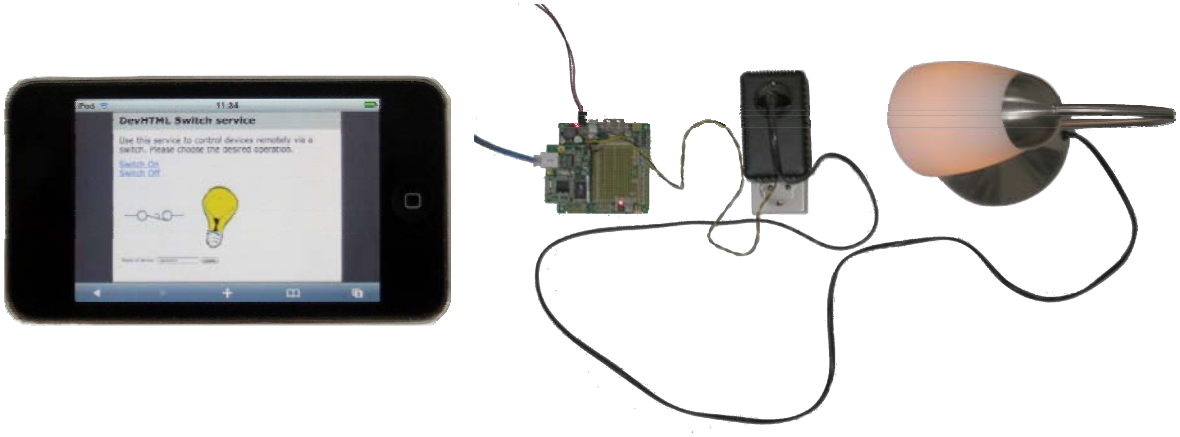


Abbildung 79 DevHTML-basierter Zugriff auf ein Prototyping Board der Firma Rabbit Semiconductor, das in Kombination mit zusätzlicher – in einem Steckergehäuse untergebrachten – Elektronik als Schalter fungiert.

Das Gerät Switch besteht aus drei Diensten, wovon *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* ausschließlich Aktionen umfasst, die allen entwickelten DevXML-Geräten gemein sind[150]. Die Aktion *string GetAdditionalName()* liefert initial den Wert „Switch 1“ zurück. Abbildung 80 zeigt das Klassendiagramm des Gerätes.

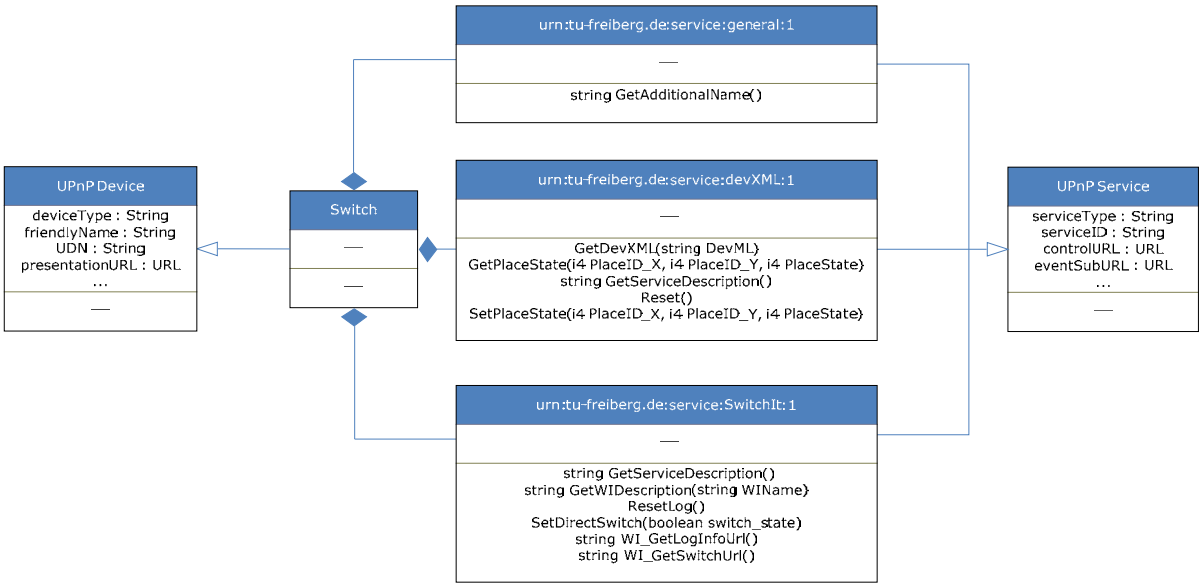


Abbildung 80 Klassendiagramm des DevControl-Gerätes Switch. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst RFID den gerätespezifischen Dienst *urn:tu-freiberg.de:service:SwitchIt:1*.

Der Dienst *urn:tu-freiberg.de:service:SwitchIt:1* wird mit *string GetServiceDescription()* näher beschrieben. Der voreingestellte Rückgabewert lautet „Switching Service“. Der Parameter *WIName* bestimmt das Ergebnis der Aktion *string GetWIDescription(string WIName)* wie in Tabelle 51 dargestellt.

[150] Vgl. hierzu Kapitel 4.2.1 „Interface von DevControl-Geräten“

Tabelle 51 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetLogInfoUrl	Access log page
WI_GetSwitchUrl	Access control page
sonst	Unknown WI

Bis zu 50 der letzten vorgenommenen Schalthandlungen werden von dem Gerät Switch mit Zeitstempel geloggt. Der Aufruf von *ResetLog()* führt zur Löschung aller bisherigen Einträge. Die Aktion *SetDirectSwitch(boolean switch_state)* bewirkt für *switch_state=true* das Öffnen, für *switch_state=false* das Schließen des Schalters. Das von *string WI_GetLogInfoUrl()* referenzierte Web Interface dient der Anzeige der geloggten Informationen. Die URL des Web Interface zur Steuerung des Schalters wird von *string WI_GetSwitchUrl()* zurückgegeben. Der DevXML-basierten Steuerung liegt das Petrinetz in Abbildung 81 zugrunde. Beachte hierzu den DevXML-Code in Anhang A2.

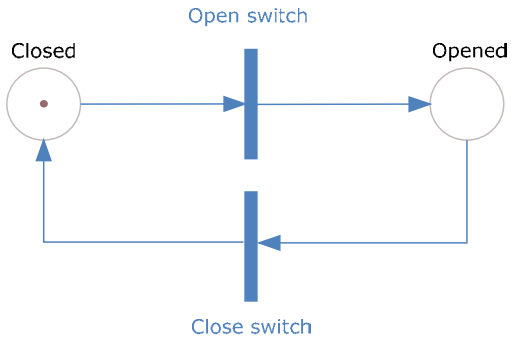


Abbildung 81 Petrinetz, das vom DevXML-Code des Gerätes Switch beschrieben wird. Die Transitionen „Open Switch“ und „Close Switch“ bewirken den Zustandsübergang von „Closed“ nach „Opened“ und umgekehrt.

Es ist der Übergang vom Zustand „Open“ in den Zustand „Closed“ und umgekehrt möglich. Die hinführenden Transitionen „Open Switch“ und „Close Switch“ greifen auf die Aktion *SetDirectSwitch(boolean switch_state)* zurück. Die Auslösung erfolgt in beiden Fällen durch Klick auf den der Transition zugeordneten Button.

6.2 Heating

Die Erweiterung eines bestehenden Gerätes um DevControl-Funktionalität ist mit geringem Aufwand möglich. Bei der realisierten, über DevControl ansprechbaren, Heizung kommt eine Steuerungseinheit der Firma Vaillant zum Einsatz. Diese lässt sich mittels USB an einen handelsüblichen PC anschließen und durch Einbindung der mitgelieferten Bibliothek via C#-Programm verwenden. Vorliegende Realisierung basiert auf einem derartigen Zusammenschluss. Abbildung 82 zeigt den Lesezugriff auf die aktuellen Temperaturvorgaben von einem iPhone unter Nutzung der DevHTML-Schnittstelle. Im Beispiel beträgt der Sollwert für die Heizwassertemperatur 60 °C, der Sollwert für die Brauchwassertemperatur 43 °C.

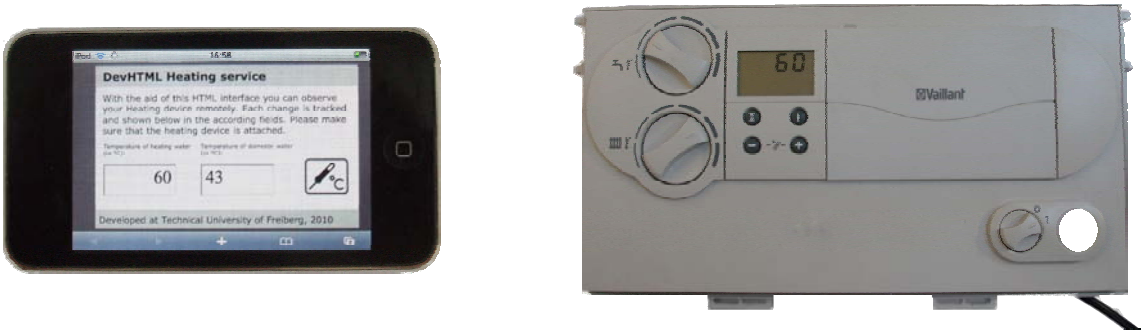


Abbildung 82 DevHTML-basierter Zugriff auf eine Steuerungseinheit der Firma Vaillant zur Ermittlung aktuell vorgegebener Heiz- und Brauchwasservorlauftemperaturen.

Aus Kostengründen wird hier auf eine Exklusivitätsbeziehung entsprechend Abbildung 76, Fall b) verzichtet. Die Windows-Anwendung Heating ist in C# entwickelt. Sie bezieht den UPnP-Stack von Intel mit ein und umfasst drei UPnP-Dienste. Nach dem Anwendungsstart kann im Feld „Name of Heating“ eine frei wählbare Bezeichnung, die mit „Heating 1“ vorbelegt ist, eingetragen werden. Der Rückgabewert von *string GetAdditionalName()* innerhalb von *urn:tu-freiberg.de:service:general:1* entspricht diesem Eintrag.

Der Dienst *urn:tu-freiberg.de:service:devXML:1* stellt die für DevXML zwingend erforderlichen – weiter oben bereits erläuterten – Aktionen bereit. Gerätespezifische Statusvariablen und Aktionen sind im Dienst *urn:tu-freiberg.de:service:heatingService:1* zusammengefasst. Siehe hierzu Abbildung 83.

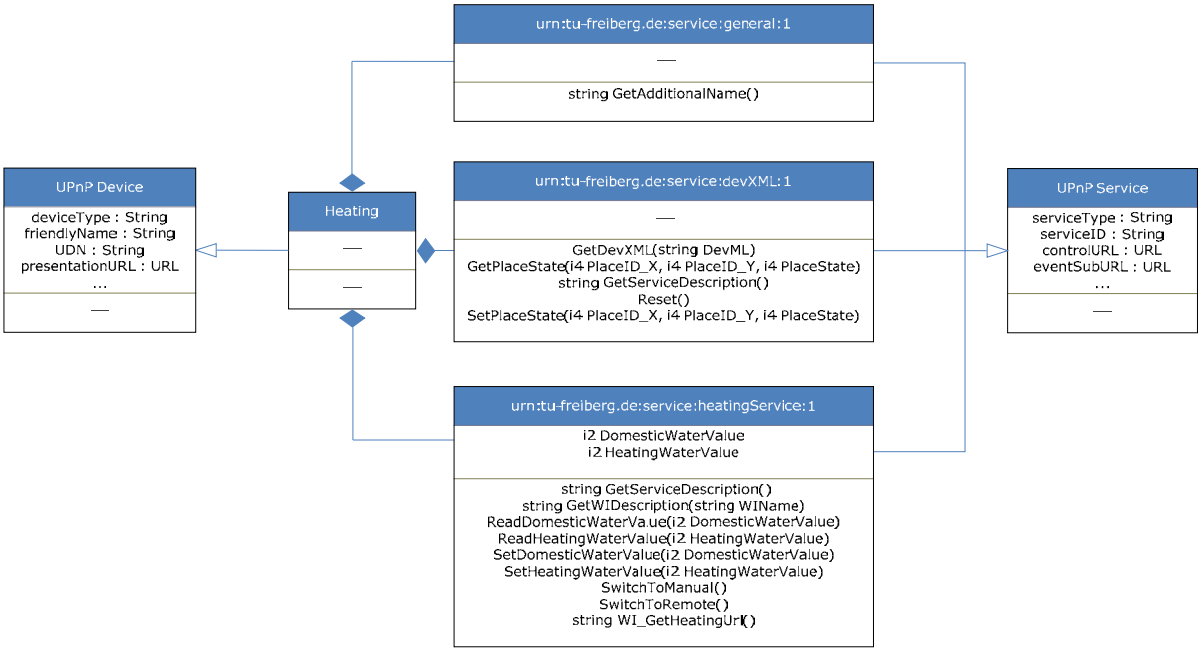


Abbildung 83 Klassendiagramm des DevControl-Gerätes Heating. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst Heating den gerätespezifischen Dienst *urn:tu-freiberg.de:service:heatingService:1*.

Die Aktion *string GetServiceDescription()* gibt die Dienstbezeichnung „Heating service“ zurück. Die möglichen Rückgabewerte der Aktion *string GetWIDescription(string WIName)* sind in Tabelle 52 aufgeführt. Mit dem Gerät Heating können die Vorlauftemperaturen von Brauch- und Heizungswasser eingestellt werden. Die Statusvariablen *DomesticWaterValue* und *HeatingWaterValue* halten die aktuellen Werte.

Die Aktionen *ReadDomesticWaterValue(i2 DomesticWaterValue)* und *ReadHeatingWaterValue(i2 HeatingWaterValue)* dienen ebenfalls der Sollwertabfrage. Das jeweilige Ergebnis wird vom Output-Parameter aufgenommen. Mit den Aktionen *SetDomesticWaterValue(i2 DomesticWaterValue)* und *SetHeatingWaterValue(i2 HeatingWaterValue)* können Temperaturvorgaben gemacht werden. Der Zielwert ist hier über den Input-Parameter zu spezifizieren.

Tabelle 52 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetHeatingUrl	Monitor values
sonst	Unknown WI

Die Steuerungseinheit lässt sich in die Modi Manuell- und Fernbetrieb versetzen. Im Manuellbetrieb sind Temperaturvorgaben direkt an den Bedienknöpfen des Vaillant-Gerätes vorzunehmen. Der Fernbetrieb zeichnet sich dadurch aus, dass Einstellungen von extern – z. B. browsergesteuert – vorgenommen werden. Mit der Aktion *SwitchToManual()* respektive *SwitchToRemote()* wird der einzunehmende Modus festgelegt. Modusunabhängig ist die Fernüberwachung der Werte stets möglich. Die URL des DevHTML-Interface ist über *WI_GetHeatingUrl()* abrufbar. Hier sind die aktuellen Vorlauftemperaturen einzusehen. Der DevXML-Code im Anhang A2 beschreibt das in Abbildung 84 visualisierte Petrinetz. Entweder ist der Platz „Manual Setting“ oder „Remote Setting“ belegt. Beide Plätze sind über die Statusvariablen *HeatingWaterValue* und *DomesticWaterValue* näher bestimmt. Der Anzeigebereich für Plätze innerhalb des DevXML-Control Point beinhaltet somit diese Charakteristika.

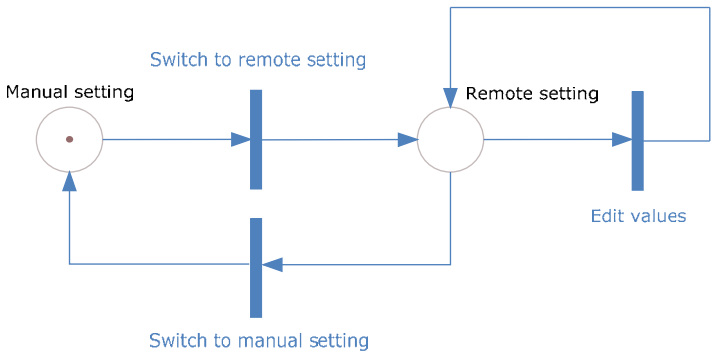


Abbildung 84 Petrinetz, das vom DevXML-Code des Gerätes Heating beschrieben wird. Das Gerät unterstützt die Modi Manuell- und Fernbetrieb. Hiermit korrespondieren die Plätze „Manual setting“ und „Remote setting“.

Der Übergang von „Manual setting“ in „Remote Setting“ ist durch Klick auf den Button *mybutton* auszulösen. Der umgekehrte Zustandsübergang vollzieht sich analog. Die Transitionsbezeichnungen lauten „Switch to remote setting“ bzw. „Switch to manual setting“.

Der Zustand „Remote setting“ zeichnet sich dadurch aus, dass vom DevXML-Control Point Vorlauftemperaturen unter Rückgriff auf Steuerelemente vom Typ *slider* gesetzt werden können. Die zugehörige Transition „Edit values“ feuert, sobald sich der Wert von *myDomesticSlider* oder *myHeatingSlider* ändert. Beim Feuern werden die dann aktuellen Werte den Aktionen *SetHeatingWaterValue(i2 HeatingWaterValue)* und *SetDomesticWaterValue(i2 DomesticWaterValue)* als Parameter übergeben. Folgezustand ist wiederum „Remote setting“.

6.3 Multimeter

Die Erweiterung des Messgerätes VC820 der Firma Voltcraft um DevControl-Funktionalität ist ein weiteres Beispiel für die Anwendbarkeit des entwickelten Konzepts auf Bestandsgeräte. Die Realisierung erfolgt als Windows-Anwendung, die aus Kostengründen wiederum auf einem nicht-exklusiv mit ihr verbundenen Standard-PC ausgeführt wird. Das Messgerät kann über eine RS232-Schnittstelle direkt mit dem PC verbunden werden. Bei Verwendung alternativer Anschlüsse wie z. B. USB-Ports sind entsprechende Adapter einzusetzen. Abbildung 85 zeigt eine typische Anordnung. Der aktuell gemessene Spannungswert von 235 V ist auf dem iPhone zu erkennen. Im Beispiel wird DevHTML genutzt.

Abbildung 85 DevHTML-basierter Zugriff auf das Messgerät VC820 zur Anzeige der aktuellen Größe bestehend aus Zahlenwert und Einheit.

Die Windows-Anwendung Multimeter wurde unter Einbeziehung des Intel UPnP-Stacks in C# entwickelt. Sie umfasst die weiter oben bereits ausführlich diskutierten Dienste *urn:tu-freiberg.de:service:general:1* sowie *urn:tu-freiberg.de:service:devXML:1*. Die Aktion *string GetAdditionalName()* liefert die Bezeichnung zurück, die im Textfeld „Name of Multimeter“ der Windows-Anwendung eingetragen ist. Per Voreinstellung lautet der Eintrag „Multimeter 1“. Des Weiteren zählt der Dienst *urn:tu-freiberg:service:measurement:1* zum Funktionsumfang. Die Statusvariablen *bin.base64 CurrentUnit* und *r8 MeasuredValue* geben Auskunft über Einheit bzw. Zahlenwert der gemessenen physikalischen Größe.

Die Aktionen *GetCurrentUnit(bin.base64 CurrentUnit)* sowie *GetMeasuredValue(r8 MeasuredValue)* können alternativ für die Größenermittlung genutzt werden. In diesem Fall ist abfrageseitig event subscription nicht erforderlich. Die Einheit wird von einem 30 Byte großen Feld beschrieben. Vergleiche hierzu Tabelle 33. Die Einträge sind base64-kodiert[151].

[151] Josefsson: The Base16, Base32, and Base64 Data Encodings, RFC 4648.

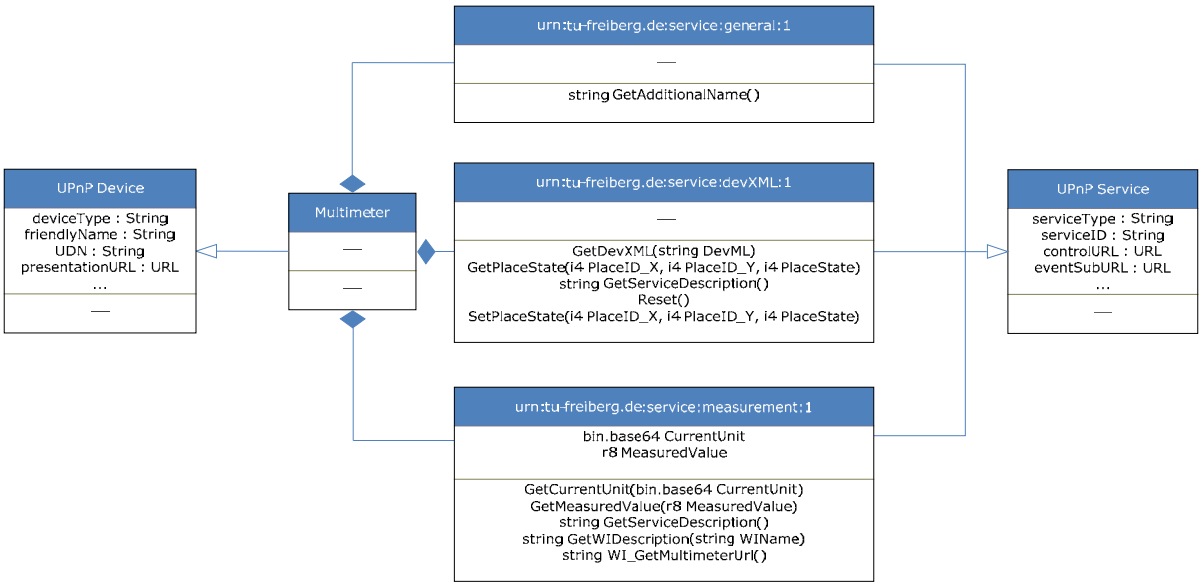


Abbildung 86 Klassendiagramm des DevControl-Gerätes Multimeter. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst Multimeter den gerätespezifischen Dienst *urn:tu-freiberg.de:service:measurement:1*.

Rückgabewert der Aktion *string GetServiceDescription()* ist „Measurement Service“. Tabelle 53 gibt Aufschluss über mögliche Ergebnisse der Aktion string *GetWIDescription(string WIName)*. Das DevControl-Gerät Multimeter verfügt über ausschließlich ein Web Interface. Die hiermit assoziierte URL ist über die Aktion *string WI_GetMultimeterUrl()* zu beziehen.

Tabelle 53 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetMultimeterUrl	Monitor values
sonst	Unknown WI

Abbildung 87 zeigt das zum DevXML-Code in Anhang A4 korrespondierende Petrinetz. Es besteht aus einem Platz. Die Mengen der Prekanten, Postkanten und Transitionen sind leer. Gemäß DevXML-Code ist dem Platz *Measurement* die Statusvariable mit dem Namen *MeasuredValue* zugeordnet. Über das Attribut *unit* wird die Variable *CurrentUnit* referenziert, die der physikalischen Größe dynamisch – entsprechend der Einstellung am Messgerät – eine Einheit in base64-Kodierung zuweist.

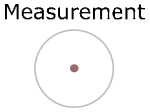


Abbildung 87 Petrinetz, das vom DevXML-Code des Gerätes Multimeter beschrieben wird. Es verfügt über ausschließlich einen Platz. In diesem erfolgt die Erfassung physikalischer Größen.

6.4 Phone

In den vorangegangenen Kapiteln wurden realisierte DevControl-Geräte diskutiert, die auf den Programmiersprachen C# und Dynamic C sowie den Geräteklassen Windows-Anwendung und Standalone Mikrocontroller basieren. Die aus Java und Mobilgerät bestehende Paarung ist ebenfalls stark vertreten (vgl. Tabelle 50).

Es folgen Ausführungen, wie unter Nutzung von J2ME die Erweiterung um DevControl-Funktionalität erfolgt. Zum Einsatz kommt ein Nokia E61, dessen Wählfunktion über DevHTML und DevXML zugänglich gemacht wird. Ein Mobilgerät eignet sich somit nicht nur als Steuergerät, sondern auch als zu steuerndes Gerät. Siehe hierzu Abbildung 88.

Abbildung 88 DevHTML-basierter Zugriff auf ein Smartphone der Firma Nokia, das unter Rückgriff auf J2ME seine Wählfunktion bereitstellt. Als Steuergerät eignet sich beispielsweise ein iPhone.

In der Praxis ist das beschriebene Anwendungsszenario von untergeordneter Relevanz. Das Gerät Phone soll vielmehr der Hinführung zu dem weiter unten beschriebenen Virtual Device Haptic Phone dienen. Die zu wählende Rufnummer ist hier auf einem RFID-Tag gespeichert. Befindet sich dieses in Reichweite eines entsprechenden Schreib-Lese-Gerätes, wird der Wählvorgang initiiert.

Die Funktionalität des DevControl-Gerätes Phone wird mit J2ME in der Konfiguration Connected Limited Device Configuration (CLDC) realisiert. Komplementär hierzu kommt das Mobile Information Device Profile (MIDP) zum Einsatz[152]. Auf dieser Grundlage basiert z. B. die HTTP-basierte Kommunikation. Der UPnP-Stapel ist eine Eigenentwicklung. Phone umfasst die in Kapitel 4.2.1 spezifizierten Dienste *urn:tu-freiberg.de:service:general:1* sowie *urn:tu-freiberg.de:service:devXML:1*. Der Dienst *urn:tu-freiberg.de:service:operator:1* stellt gerätespezifische Aktionen bereit. Siehe Abbildung 89.

[152] Topley: J2ME in a Nutshell, S. 8 ff.

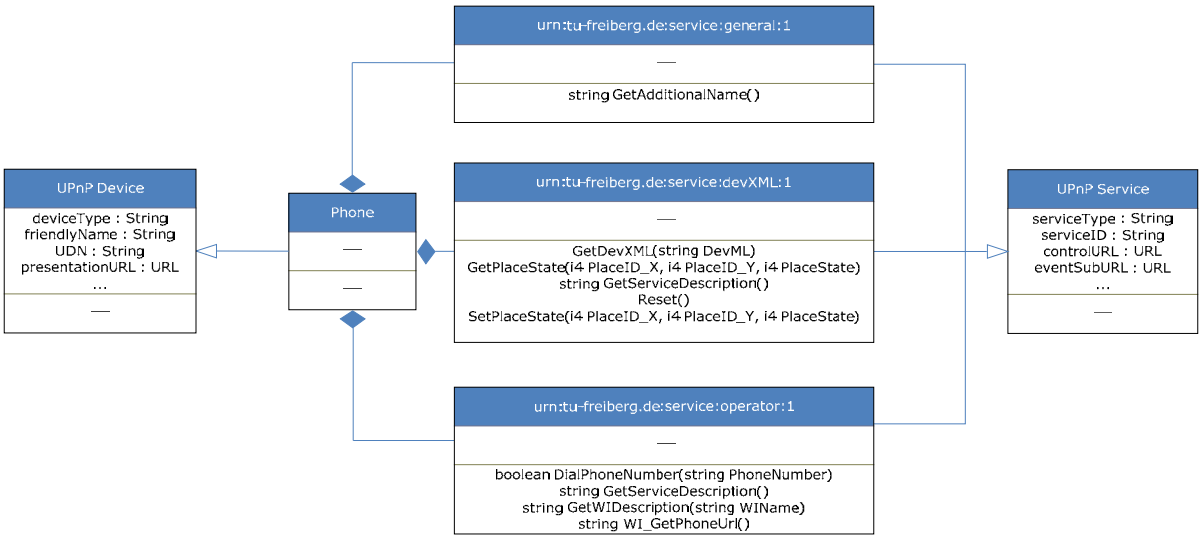


Abbildung 89 Klassendiagramm des DevControl-Gerätes Phone. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst Phone den gerätespezifischen Dienst *urn:tu-freiberg.de:service:operator:1*.

Die Aktion *string GetAdditionalName()* des Dienstes *urn:tu-freiberg.de:service:general:1* liefert den aktuellen Gerätenamen zurück. Innerhalb der J2ME-Applikation kann hierfür eine beliebige Zeichenkette angegeben werden. Im Hauptmenü ist der Eintrag „More“ und im folgenden Untermenü der Eintrag „Change device name“ zu wählen.

Mit Aufruf von *boolean DialPhoneNumber(string PhoneNumber)* wird die Wahl der Rufnummer *PhoneNumber* eingeleitet. Der Rückgabewert *true* signalisiert die erfolgreiche Funktionsausübung. Tritt ein Fehler auf, ist der Rückgabewert *false* die Folge. Die Aktion *string GetServiceDescription* liefert als Ergebnis die Dienstbezeichnung „Operator service“. Tabelle 54 zeigt die möglichen Rückgabewerte der Aktion *string GetWIDescription(string WIName)*.

Tabelle 54 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetPhoneUrl	Dial number
sonst	Unknown WI

Der Link des Web Interface mit der Beschreibung „Dial number“ kann durch Aufruf von *string WI_GetPhoneUrl()* ermittelt werden. Die dazugehörende Seite befähigt zur Rufnummernwahl von einem Steuergerät auf Grundlage von DevHTML. Des Weiteren unterstützt Phone die Steuerung mittels DevXML. Abbildung 90 zeigt das Petrinetz, das aus dem assoziierten Code in Anhang A5 resultiert.

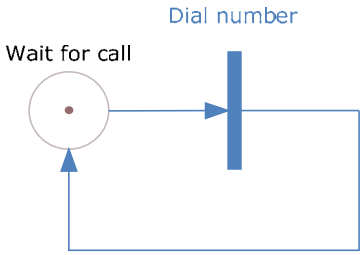


Abbildung 90 Petrinetz, das vom DevXML-Code des Gerätes Phone beschrieben wird. Die Rufnummernwahl ist stets einleitbar. „Wait for call“ stellt den Ausgangs- und Zielzustand der Transition „Dial number“ dar.

Es verfügt über den Platz „Wait for call“, der permanent belegt ist. Jederzeit kann somit vom DevXML-Control Point die Rufnummernwahl initiiert werden. Ein Klick auf den Button *myButton* bewirkt die Feuerung der Transition „Dial number“. Hiermit verbunden ist der Aufruf der Aktion *boolean DialPhoneNumber(string PhoneNumber)*, deren Parameter den Wert des Eingabefeldes *myNumberBox* annimmt.

6.5 Terminal

Die Anwendung Terminal ermöglicht die Fernbedienung und -beobachtung des Computers, auf dem sie ausgeführt wird. Hierzu übermittelt sie in regelmäßigen Abständen die Bildschirminhalte an ein Steuergerät. Dieses wiederum registriert Tastatureingaben sowie Mausereignisse, die dann rückgemeldet werden. Auf diese Weise sind jegliche PC-basierte Steuerhandlungen gleichermaßen vor Ort und aus der Ferne durchführbar. In ihrer Funktionsweise ist die Anwendung DevHTML-konform. Abbildung 91 zeigt eine USB-Steuerleiste, die über ein Smartphone der Firma Nokia konfiguriert wird.

Abbildung 91 Die DevHTML-basierte Anwendung Terminal wird verwendet, um auf ein nicht DevControl-konformes Gerät zuzugreifen. Im Beispiel wird eine USB-Steckdosenleiste der Firma Gembird mit einem Smartphone konfiguriert. Unter Rückgriff auf die Software des Herstellers werden die Steckdosen 1 und 2 aktiviert.

Im Ergebnis sind von den vier per Software steuerbaren Steckdosen – erkennbar an den orangefarbenen Statusleuchten – die beiden links befindlichen aktiviert. Im Gegensatz zur direkten Steuerung mittels DevHTML oder DevXML wird bei Verwendung von Terminal zwingend ein PC mit dem Betriebssystem Windows benötigt, der dem zu steuernden Gerät zuzuordnen ist. Ferner eignet sich Terminal nicht bei niedrigen Datenübertragungsgeschwindigkeiten, da die kontinuierliche Übermittlung von Bildschirminhalten ressourcenintensiv ist. Durch die Beschränkung auf steuerungsrelevante Statusinformationen wie „On“ und „Off“ kann die Netzwerklast hingegen erheblich reduziert werden.

Bei manchen Handybetriebssystemen führt ein Doppelklick nicht zum JavaScript-Ereignis *DblClick*. Stattdessen wird zweimal in Folge das Ereignis *Click* ausgelöst. Die Anwendung Terminal trägt diesem Umstand Rechnung, indem die Behandlung zweier aufeinanderfolgender Ereignisse vom Typ *Click* mit der eines Ereignisses vom Typ *DblClick* übereinstimmt. Voraussetzung ist ein zeitlicher Abstand kleiner 300 ms.

Die Inter-Gerätekommunikation ist auf Basis von Terminal nur schwer umsetzbar, da Zustände und ausführbare Aktionen ausschließlich aus visuellen Informationen hervorgehen. Überdies werden kontextsensitive Bedienkonzepte wie z. B. das Einblenden einer virtuellen Tastatur bei Aktivierung von Eingabefeldern nicht unterstützt. DevXML mit der formalisierten Gerätebeschreibung unterliegt diesen Einschränkungen nicht. Aus vorgenannten Punkten wird deutlich, dass Terminal keinesfalls DevHTML oder DevXML ersetzen kann. Eine typische Anwendung von Terminal ist die Integration von Geräten, die weder die Vorzüge von DevHTML noch DevXML aufweisen. Somit kann deren Kernfunktionalität verfügbar gemacht werden. Die Anwendung umfasst die Dienste *urn:tu-freiberg.de:service:general:1* sowie *urn:tu-freiberg.de:service:remote:1*, die mit ihren Aktionen in Abbildung 92 aufgeführt sind.

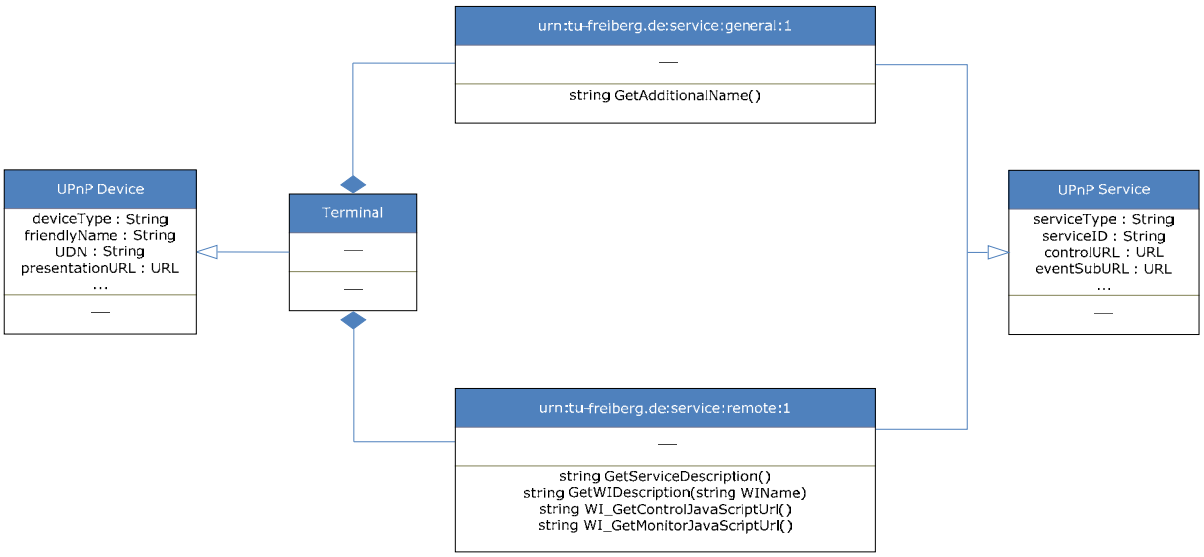


Abbildung 92 Klassendiagramm des DevControl-Gerätes Terminal. Neben dem allgemeinen Dienst *urn:tu-freiberg.de:service:general:1* gemäß Kapitel 4.2.1 umfasst Terminal den gerätespezifischen Dienst *urn:tu-freiberg.de:service:remote:1*.

Von *string GetAdditionalName()* innerhalb von *urn:tu-freiberg.de:service:general:1* wird die Zeichenkette „Terminal 1“ zurückgeliefert. Alle weiteren Aktionen sind zum Dienst *urn:tu-freiberg.de:service:remote:1* gehörig. Der Rückgabewert von *string GetServiceDescription()* lautet „Remote service“. Die Beschreibungen der Web Interfaces lassen sich mit *string GetWIDescription(string WIName)* ermitteln. Der zwischen dem Parameter *WIName* und dem Ergebnis des Aktionsaufrufes geltende Zusammenhang ist in Tabelle 55 dargestellt.

Tabelle 55 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetControlJavaScriptUrl	Remote Controlling
WI_GetMonitorJavaScriptUrl	Remote Monitoring
sonst	Unknown WI

Die URLs der Benutzerschnittstellen für die Fernbedienung respektive -beobachtung sind über die Aktionen *string WI_GetControlJavaScriptUrl()* sowie *WI_GetMonitorJavaScriptUrl()* zu beziehen. Die korrekte Darstellung der Seiteninhalte bedingt die browserseitige Aktivierung von JavaScript.

6.6 Jukebox

Mit der in C# entwickelten Anwendung Jukebox ist der ortsungebundene Zugriff auf Mediendaten möglich. Jukebox setzt auf UPnP AV (Audio and Video) auf. Die Abgrenzung von dieser Basisarchitektur erfolgt durch zusätzliche Unterstützung von DevHTML und DevXML. Während der rein UPnP AV-basierte Ansatz für Steuerungszwecke sogenannte UPnP AV-Controller bedingt, sind für die Bedienung von Jukebox beliebige Endgeräte mit Browser ausreichend. Hierbei ist sicherzustellen, dass im Netzwerk mindestens ein UPnP AV-Server verfügbar ist. Als solcher eignet sich beispielsweise ein Router des Typs WRT160NL des Herstellers Cisco. Dieser bezieht zu publizierende Mediendaten von einem angeschlossenen USB-Stick.

Für die Wiedergabe von Inhalten ist entweder das Steuergerät selbst oder ein im Netzwerk befindlicher UPnP AV-Renderer auszuwählen. Abbildung 93 zeigt zuletzt genannten Fall. Mit einem iPhone wird als Medienquelle ein Router des oben bezeichneten Typs und als Abspielgerät ein PC mit Windows Media Player 12[153] ausgewählt. Es folgt die Selektion des Musikstückes „Sleep away“ sowie die Initiierung des Abspielvorgangs.

[153] Alternativ kann die Anwendung Intel UPnP AV Renderer verwendet werden.

Abbildung 93 DevHTML-basierte Steuerung einer Jukebox. Als Medienquelle wird ein Cisco-Router des Typs WRT160NL und als Abspielgerät ein PC mit Windows Media Player 12 selektiert. Die Wiedergabe des dann ausgewählten Musikstückes „Sleep away“ schließt sich an. Alternativ erfolgt sie unmittelbar auf dem Steuergerät.

Die Anwendung Jukebox ist in C# entwickelt. Sie schließt den UPnP-Stapel der Firma Intel mit ein und setzt sich aus den drei Diensten *urn:tu-freiberg.de:service:general:1*, *urn:tu-freiberg.de:service:devXML:1* und *urn:tu-freiberg.de:service:music:1* zusammen. Ausschließlich zuletzt genannter ist über gerätespezifische Methoden definiert. Siehe hierzu Abbildung 94.

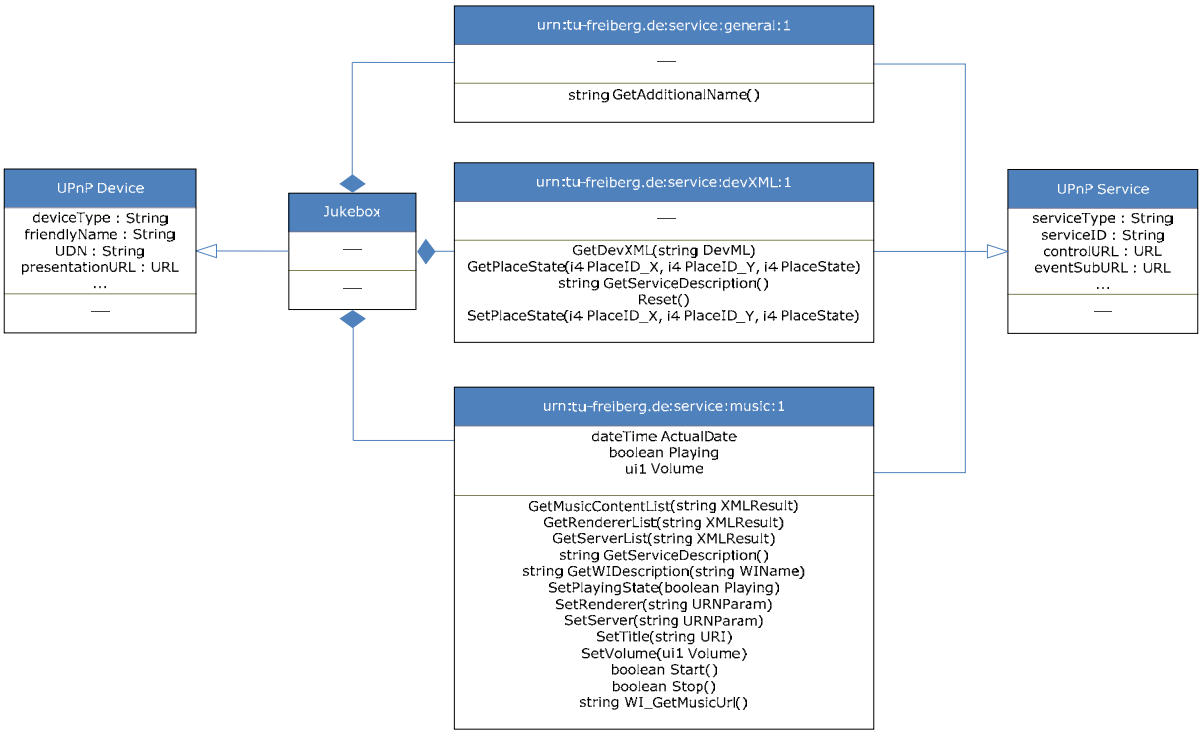


Abbildung 94 Klassendiagramm des DevControl-Gerätes Jukebox. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst Jukebox den gerätespezifischen Dienst *urn:tu-freiberg.de:service:music:1*.

Die bereits mehrfach erwähnte Methode *string GetAdditionalName()* des Dienstes *urn:tu-freiberg.de:service:general:1* liefert die Zeichenkette zurück, die im Feld „Name of Jukebox“ der Anwendung eingetragen ist. Die Vorbelegung lautet „Jukebox 1“.

Zum Dienst *urn:tu-freiberg.de:service:music:1* zählen die drei Statusvariablen *dateTime ActualDate*, *boolean Playing* und *ui1 Volume*. Sie geben Auskunft über das aktuelle Datum inklusive Uhrzeit, den geltenden Wiedergabestatus sowie die Wiedergabelautstärke. Die in Kapitel 4.3.1 „Datenmodell“ eingeführte Datenstruktur *table* wird vom output-Parameter *XMLResult* der Methoden *GetMusicContentList(string XMLResult)*, *GetRendererList(string XMLResult)* und *GetServerList(string XMLResult)* eingehalten. Entsprechend den Bezeichnungen beinhalten die einspaltigen Ergebnismatrizen Angaben zu verfügbaren Medientiteln, -renderern und -servern. Die nähere Bezeichnung des Dienstes – vorliegend „Music Service“ – ist über die Methode *string GetServiceDescription()* abrufbar. Die dem oben abgebildeten Webinterface zugeordnete Beschreibung ist mit der Aktion *string GetWIDescription(string WIName)* ermittelbar. Es gilt die Parameterabhängigkeit gemäß Tabelle 56. Neben „Playback“ existieren keine weiteren Webinterfaces.

Tabelle 56 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetMusicUrl	Playback
sonst	Unknown WI

Mit der Aktion *SetPlayingState(boolean Playing)* kann der Wert der Statusvariablen *Playing* gesetzt werden. Es ist zu beachten, dass bei der direkten Manipulation die notwendigen Vorkehrungen für das Erreichen des entsprechenden Gerätezustandes zu treffen sind. In DevXML wird dies mit der variablenabhängigen Feuerung von Transitionen bewerkstelligt. Beispielsweise erfordert das Setzen von *Playing=false* den nachgelagerten Aufruf der Aktion *boolean Stop()*.

In DevXML gestatten Statusvariablen die Durchführung von Fallunterscheidungen, die eingabeabhängig die Petrinetzmarkierungen beeinflussen. Es ist möglich, innerhalb des User Interface einer Transition beliebig viele Statusvariablen zu modifizieren. Von deren Werten ist dann abhängig, welche weiteren Transitionen zu feuern sind[154].

Die Selektion des zu verwendenden UPnP AV-Renderers und -Servers ist mit den Aktionen *SetRenderer(string URNParam)* sowie *SetServer(string URNParam)* vorzunehmen. Mit *URNParam* wird die jeweilige Komponente über ihren Unique Resource Name spezifiziert. Den von UPnP AV-Servern bereitgestellten Mediendateien sind Unique Resource Identifier zugewiesen. Diese eindeutigen Bezeichner sind im Rahmen der Titelauswahl unter Verwendung der Aktion *SetTitle(string URI)* als Parameter zu übergeben.

Für die Festsetzung der Wiedergabelautstärke ist *SetVolume(ui Volume)* aufzurufen. Der gültige Wertebereich geht von 0 bis 100. Der Abspielvorgang wird mit *boolean Start()* eingeleitet und mit *boolean Stop()* beendet. Bei fehlerfreier Funktionsausübung wird *true*, andernfalls *false* zurückgegeben.

[154] Ohne den Einsatz von Statusvariablen geht ein Petrinetz in die durch die gefeuerte Transition beschriebene Markierung über. Zum Feuern eingabeabhängiger Folgetransitionen kommt es nicht.

Die URL des weiter oben bereits angeführten Webinterface „Playback“ liefert die Aktion *string WI_GetMusicUrl()*. Der zur Jukebox gehörige DevXML-Code in Anhang A6 beschreibt das Petrinetz gemäß Abbildung 95. Die Bedeutung der einzelnen Plätze und Transitionen wird nachfolgend aufgezeigt.

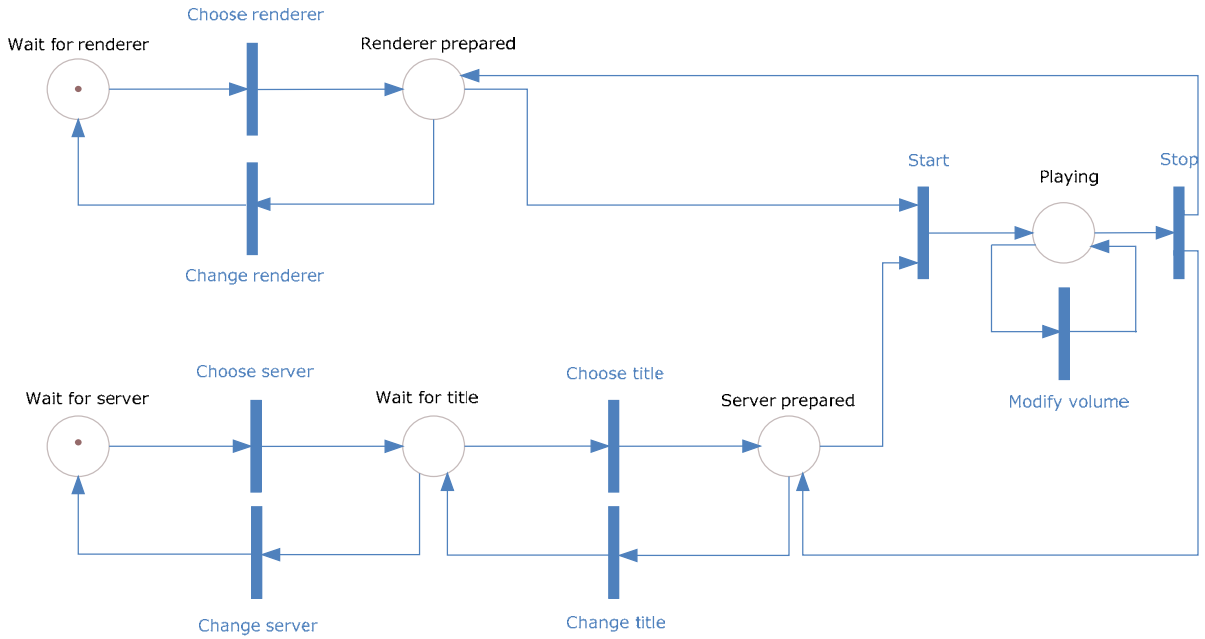


Abbildung 95 Petrinetz, das vom DevXML-Code des Gerätes Jukebox beschrieben wird. Der Zustand „Playing“ kennzeichnet die Wiedergabe einer Mediendatei. Er ist zu erreichen, indem ausgehend von der dargestellten Belegung Medienrenderer, -server und -titel ausgewählt werden.

Initial sind die Plätze „Wait for renderer“ und „Wait for server“ belegt. Es ist ersichtlich, dass hieraus die Transitionen „Choose renderer“ und „Choose server“ gefeuert werden können. Microsoft Expression Blend diente der Erstellung der zugehörigen User Interfaces, die infolge des auf *true* gesetzten Attributs *externalUI* des betreffenden <*transition*>-Elements einbezogen werden. Alle User Interfaces beinhalten die vom jeweiligen <*transitioncontrol*>-Element umschlossenen Kontrollelemente. Bei „Choose renderer“ und „Choose server“ sind diese vom Typ *table* sowie *button*.

Wie in Kapitel 4.3.2.3 „Elemente zur Transitionsspezifikation“ dargelegt, korrespondiert der DevXML-Steuerelementtyp *table* mit dem Silverlight-Steuerelementtyp *DataGrid* (vgl. Tabelle 37). Aus diesem Grund muss das externe User Interface über ein *DataGrid*-Objekt verfügen, das jedoch nicht zwingend für die Interaktion mit dem Nutzer heranzuziehen ist. Generell besteht die Möglichkeit, hierfür ein alternatives Steuerelement einzusetzen. In diesem Fall ist sicherzustellen, dass die eingebundene – im kompilierten XAP-Format vorliegende – Silverlight-Projektdatei ebenfalls die Programmlogik für die beidseitig gerichtete Synchronisation zwischen den Objekten integriert. Im User Interface für die Auswahl von Medienrenderer und -server sind die *DataGrid*-Objekte ausgeblendet. Gegenüber dem Anwender treten ausschließlich Objekte vom Typ *C1CoverFlow*[155] in Erscheinung. Der erforderliche Statusabgleich wird mit Ereignishandlern realisiert. Siehe Abbildung 96.

[155] Entwicklung der Firma ComponentOne.

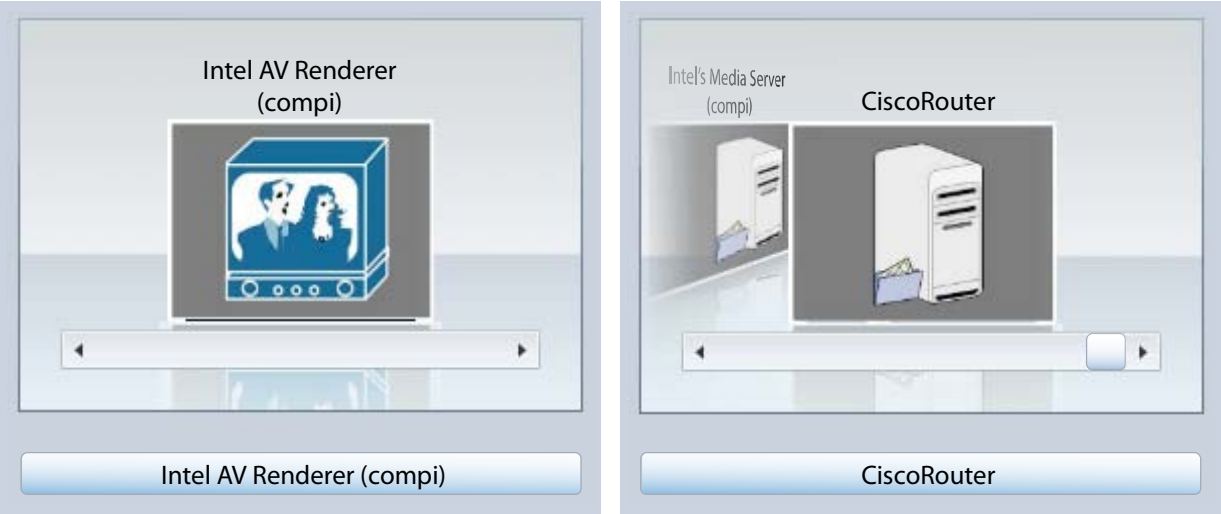


Abbildung 96 User Interfaces für die Auswahl von Medienrenderer und -server. Das Design der Kontrollelemente vom Typ *table* und *button* wurde mit dem Programm Expression Blend vorgenommen.

Im zeitlichen Ablauf folgt der Selektion des Medienservers die des Medientitels mittels „Choose title". Nach dem Treffen der vorgenannten renderer- und serverspezifischen Einstellungen sind die Plätze „Renderer prepared" sowie „Server prepared" belegt. Ausgehend hiervon ist die Transition „Start" feuerbar. Sie überführt das Netz in den Zustand „Playing", in dem sowohl die Wiedergabe erfolgt als auch die ihm zugeordneten Statusvariablen *ActualDate* sowie *Volume* im DevXML-Control Point zur Anzeige gebracht werden. Beachte hierzu den zugehörigen <*place*>-Abschnitt.

Das User Interface der Transition „Modify volume" befähigt zur Lautstärkeregelung. Nutzereingaben resultieren in sich ändernden Werten der Statusvariablen *Playing*. Hierauf basiert ebenfalls das Wirkprinzip der Transition „Stop". Im Abschnitt <*transitionaction*> ist dessen Subelement <*firecondition*> der Typ *VariableState* zugewiesen. In Kombination mit den weiteren Parametrierungen wird festgelegt, dass die Transition feuert, sobald *Playing* den booleschen Wert *false* annimmt. Die Transitionen „Change renderer", „Change server" und „Change title" sind dazu geeignet, Änderungen in Bezug auf Medienrenderer, -server und -titel vorzunehmen.

Wie in Kapitel 4.3.4 erörtert, kann unter Anwendung des Invariantenkalküls Modellzuverlässigkeit gewährleistet werden. Bei dem Gerät Jukebox wird eine demgemäße Prüfroutine direkt in die Methode *SetPlaceState(i4 PlaceID_X, i4 PlaceID_Y, i4 PlaceState)* eingebettet und damit bei jedem Zustandsübergang ausgeführt. Die Ergebnismenge des Algorithmus von Martinez umfasst die P-Invarianten $\Theta_{P1} = (1 \;\; 1 \;\; 1 \;\; 0 \;\; 0 \;\; 0)^T$ und $\Theta_{P2} = (0 \;\; 0 \;\; 1 \;\; 1 \;\; 1 \;\; 1)^T$. Zufolge Satz 9 ergibt deren lineare Überlagerung eine P-Invariante mit maximaler Netzabdeckung. Die innerhalb der Prüfroutine verwendete P-Invariante $\Theta_{P3} = \Theta_{P1} + \Theta_{P2} = (1 \;\; 1 \;\; 2 \;\; 1 \;\; 1 \;\; 1)^T$ erfüllt somit diese Eigenschaft. Hierbei entsprechen die Wichtungen in der Reihenfolge ihrer Nennung den Plätzen „Wait for renderer", „Renderer prepared", „Playing", „Wait for server", „Wait for title" und „Server prepared". Alle gültigen Zustände zeichnen sich durch eine konstante Markensumme von 2 aus. Bei Abweichung hiervon wird auf einen nicht ordnungsgemäßen Zustandsübergang geschlossen. In der Folge nimmt Jukebox den Initialzustand $M_0 = (1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0)^T$ an. In diesem sind die Plätze „Wait for renderer" sowie „Wait for server" markiert.

6.7 Alarm

Mit der Anwendung Alarm ist die zeitabhängige Auslösung von Ereignissen realisierbar. Wird Alarm alleine – d. h. nicht im Kontext eines virtuellen Gerätes – ausgeführt, zeichnet sich der Alarmzustand durch ein Akustiksignal sowie die auf *true* gesetzte Statusvariable *AlarmActive* aus. Andernfalls sind auch komplexere Schalthandlungen möglich (vgl. Kapitel 6.8.2 „Automated Switch“).

Die mit „Alarm 1“ vorbelegte Gerätebezeichnung kann im Textfeld „Name of alarm“ frei gewählt werden. Des Weiteren ist die Vorgabe von bis zu drei Zeitintervallen im Gruppenrahmen „Alarm settings“ möglich. Sie bestimmen, wann der Alarmzustand eingenommen wird. Mit den rechtsseitig angeordneten Checkboxes erfolgt deren Aktivierung bzw. Deaktivierung. Die Anwendung umfasst die drei Dienste *urn:tu-freiberg.de:service:general:1*, *urn:tu-freiberg.de:service:devXML:1* und *urn:tu-freiberg.de:service:information:1*.

Die Aktion *string GetAdditionalName()* des Dienstes *urn:tu-freiberg.de:service:general:1* liefert die unter „Alarm Name“ eingetragene Zeichenkette zurück. Der bereits mehrfach implementierte Dienst *urn:tu-freiberg.de:service:devXML:1* ist allgemeiner Natur, er wird in Kapitel 4.2.1 behandelt. Abbildung 99 zeigt das Klassendiagramm.

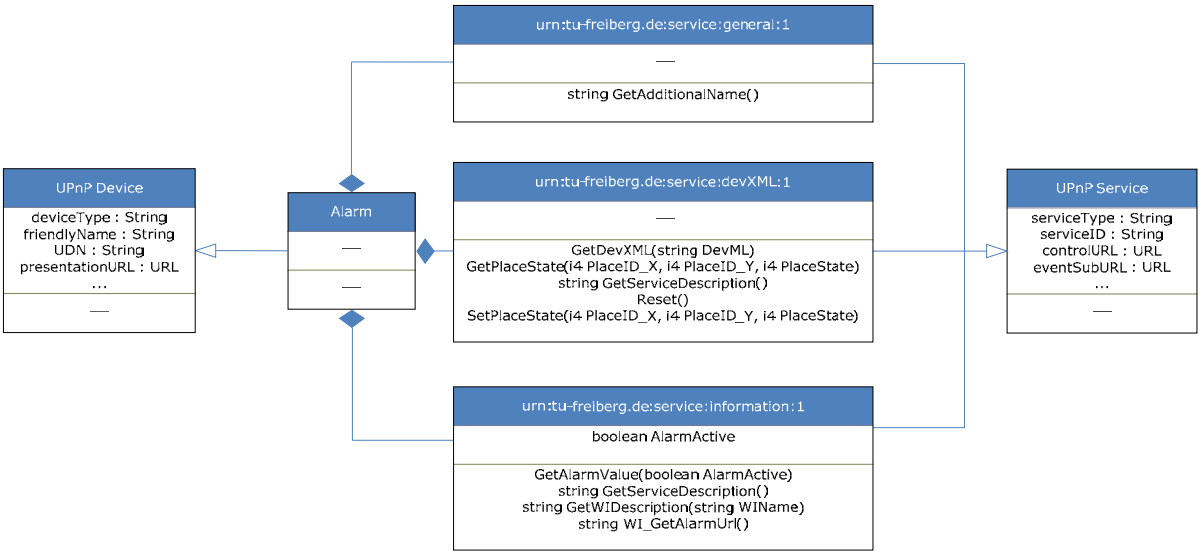


Abbildung 97 Klassendiagramm des DevControl-Gerätes Alarm. Neben den allgemeinen Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* gemäß Kapitel 4.2.1 umfasst Alarm den gerätespezifischen Dienst *urn:tu-freiberg.de:service:information:1*.

Die Statusvariable *AlarmActive* innerhalb des Dienstes *urn:tu-freiberg.de:service:information:1* nimmt im Alarmzustand den booleschen Wert *true*, sonst den booleschen Wert *false* an. Der aktuelle Status kann durch Ereignisregistrierung oder expliziten Aufruf der Aktion *GetAlarmValue(boolean AlarmActive)* ermittelt werden. Die von *string GetServiceDescription()* zurückgelieferte Dienstbeschreibung lautet „Information service“. Das Ergebnis des Aktionsaufrufes *string GetWIDescription(string WIName)* ist parameterabhängig. Tabelle 57 stellt die Zusammenhänge dar. Das über *string WI_GetAlarmUrl()* referenzierte Webinterface trägt die Bezeichnung „Define settings“. Alarmeinstellungen können im Webinterface sowie auf die oben beschriebene Weise direkt im Programm vorgenommen werden.

Tabelle 57 Rückgabewerte der Aktion *string GetWIDescription(string WIName)* in Abhängigkeit des Parameters *WIName*.

WIName	Rückgabewert
WI_GetAlarmUrl	Define settings
sonst	Unknown WI

Der DevXML-Code in Anhang A7 beschreibt das Petrinetz gemäß Abbildung 81. Die Transition „Set alarm off“ feuert, wenn ausgehend von „Alarm on“ die Statusvariable *AlarmActive* den booleschen Wert *false* annimmt. Aus der Folgemarkierung „Alarm off“ heraus feuert „Set alarm on“ beim Wahrheitswert *true*. Hiermit verbunden ist die Rücküberführung in „Alarm on“.

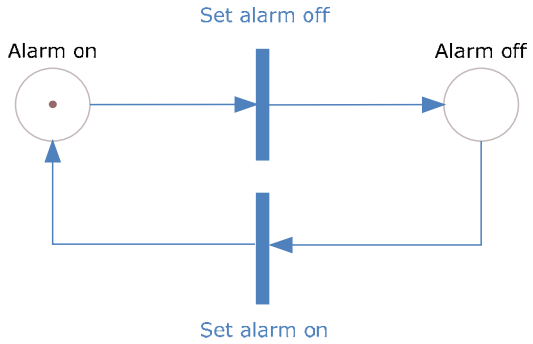


Abbildung 98 Petrinetz, das vom DevXML-Code des Gerätes Alarm beschrieben wird. Die Transitionen „Set alarm off“ und „Set alarm on“ bewirken den Zustandsübergang von „Alarm on“ nach „Alarm off“ und umgekehrt.

6.8 Virtual Devices

Die Zusammenführung geräteübergreifender Teilfunktionalitäten liegt dem Konzept der Virtual Devices zugrunde. Innerhalb von DevXML gestattet die Petrinetz-basierte Modellierung die Referenzierung von Aktionen und Statusvariablen bereits bestehender Geräte. Mit wenig Aufwand lassen sich so neue Funktionen realisieren. Die Anwendung Virtual Device erfüllt DevXML-Konformitätsanforderungen gemäß Kapitel 4.2.1. Der von ihr zu publizierende DevXML-Code ist frei bestimmbar. Hierzu muss nach Klick auf den Button „Change“ in dem sich öffnenden Auswahldialog die als Quelle dienende Datei angegeben werden. Das Textfeld „Device Name“ beinhaltet den zu verwendenden Gerätenamen. Voreingestellt ist die Zeichenkette „Virtual Device 1“. Abbildung 99 zeigt das Klassendiagramm mit den beiden Diensten *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1*. Die zum erstgenannten Dienst zählende Aktion *string GetAdditionalName()* liefert den im Textfeld „Device name“ eingetragenen Gerätenamen zurück.

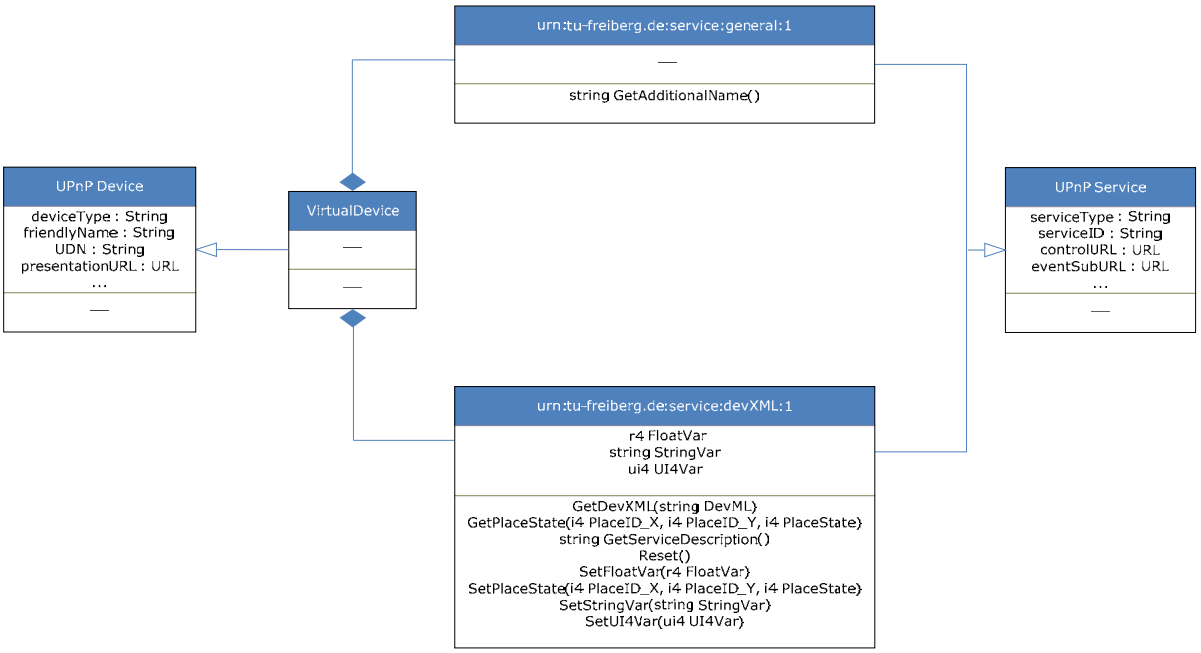


Abbildung 99 Klassendiagramm der Anwendung Virtual Device, die ausschließlich die beiden allgemeinen Dienste *urn:tu-freiberg.de:service:general:1* und *urn:tu-freiberg.de:service:devXML:1* mit einbezieht. Der über *GetDevXML(string DevXML)* abfragbare DevXML-Quellcode kann frei bestimmt werden.

Der Dienst *urn:tu-freiberg.de:service:devXML:1* schließt ergänzend zur Schnittstellenvorgabe die generischen Statusvariablen *r4 FloatVar*, *string StringVar* und *ui4 UI4Var* mit ein. Die Aktionen *SetFloatVar(r4 FloatVar)*, *SetStringVar(string StringVar)* und *SetUI4Var(ui4 UI4Var)* ermöglichen die Zuweisung von Werten. Hiermit können Ergebnisse von Aktionen, die bei den referenzierten Geräten selbst keinen Einfluss auf Platzbelegungen haben und daher nicht als spezifische Statusvariablen ausgeprägt sind, im neu geschaffenen Gerätekontext bei der bedingten Feuerung von Transitionen Berücksichtigung finden. Hinsichtlich der verbleibenden Aktionen des Dienstes sei wiederum auf Kapitel 4.2.1 verwiesen. Konkrete Anwendungsbeispiele werden mit „Haptic Phone“ und „Automated Switch“ im Folgenden vorgestellt.

6.8.1 Haptic Phone

Haptic Phone ermöglicht die Rufnummernwahl unter Umgehung des Tastenfeldes. Kontrastiv sind für den Wählvorgang RFID-Tags zu verwenden, die jeweils mit einer Rufnummer vorbelegt sind. Bei Einsatz von Tags unterschiedlicher Geometrien kann die zu wählende Rufnummer somit anhand haptisch zu erfassender Merkmale wie Rundungen, Ecken etc. bestimmt werden. Diese Verfahrensweise stellt beispielsweise für Personen mit Seheinschränkungen einen Vorteil dar. Das Klassendiagramm in Abbildung 99 ist für Haptic Phone – ebenso wie für jedes andere virtuelle Gerät – gültig.

Haptic Phone bezieht die weiter oben beschriebenen DevControl-Geräte RFID und Phone mit ein. Die Bezugnahme erfolgt im zugehörigen DevXML-Code (siehe Anhang A8), der innerhalb der Anwendung Virtual Device als Quelle anzugeben ist. Das hierin beschriebene Petrinetz zeigt Abbildung 100.

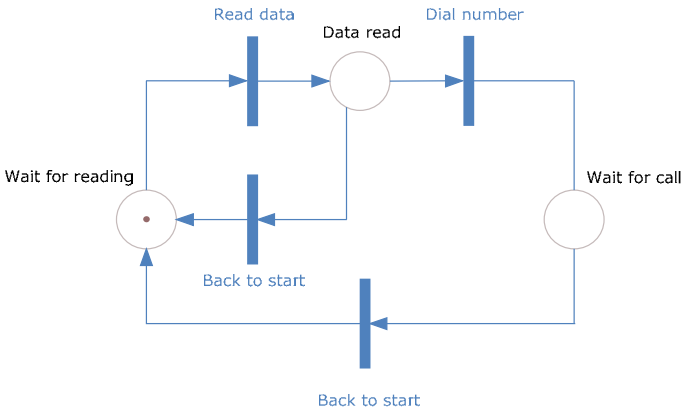


Abbildung 100 Petrinetz, das vom DevXML-Code des Gerätes Haptic Phone beschrieben wird. „Wait for reading“ stellt den Initialzustand dar.

Initial ist der Platz „Wait for reading“ belegt. Hiervon ausgehend feuert die Transition „Read Data“ automatisch, da das assoziierte Element *<firecondition>* vom Typ *always* ist. Im Rahmen der Feuerung werden vom RFID-Schreib-Lese-Gerät mit der Bezeichnung „RFID 1“ aktuelle Tag-Daten angefordert. Befindet sich kein Tag in Reichweite wird ein Leerstring zurückgeliefert, andernfalls die auf dem Tag gespeicherte Rufnummer. Die Statusvariable *StringVar* nimmt das Ergebnis auf.

In der Folgemarkierung „Data read“ kommt es zur Fallunterscheidung. Ist *StringVar* ein Leerstring zugewiesen, führt dies zur Feuerung von „Back to start“ und damit zur Überführung des Netzes in den Initialzustand. Die Transition „Dial number“ feuert hingegen, wenn *StringVar* eine nichtleere Zeichenfolge repräsentiert. Hierbei wird *StringVar* der Aktion *boolean DialPhoneNumber(string PhoneNumber)* des mit „Phone 1“ bezeichneten Gerätes als Parameter übergeben. Damit startet der Wählvorgang. Im Folgezustand „Wait for call“ verharrt Haptic Phone solage, bis durch Klick auf den Button *myButton* die Transition „Back to start“ feuert. Im Ergebnis wird die Ursprungsbelegung „Wait for reading“ eingenommen.

6.8.2 Automated Switch

Anhand des virtuellen Gerätes Automated Switch wird die zeit- und helligkeitsabhängige Automatisierbarkeit demonstriert. Während Switch als Einzelgerät manuell zu bedienen ist, gehen Schalthandlungen in der hier vorliegenden Konfiguration von den DevControl-Geräten Alarm und Webcam aus.

Überdeckt eines der drei mit der Anwendung Alarm einstellbaren Intervalle die aktuelle Uhrzeit, liegt an *V1* die Spannung 230 V an. Siehe auch Abbildung 78. Gleiches gilt bei Dämmerzustand, der durch Unterschreitung eines vorgegebenen mittleren Rotwertes des von der Webcam erfassten Bildes signalisiert wird. Im Beispiel ist der Schwellwert auf 100 gesetzt. Wiederum beschreibt das in Kapitel 6.8 dargestellte Klassendiagramm die unmittelbaren von Automated Switch bereitgestellten Dienste und Aktionen. Der DevXML-Code in Anhang A9 korrespondiert mit dem Petrinetz in Abbildung 101.

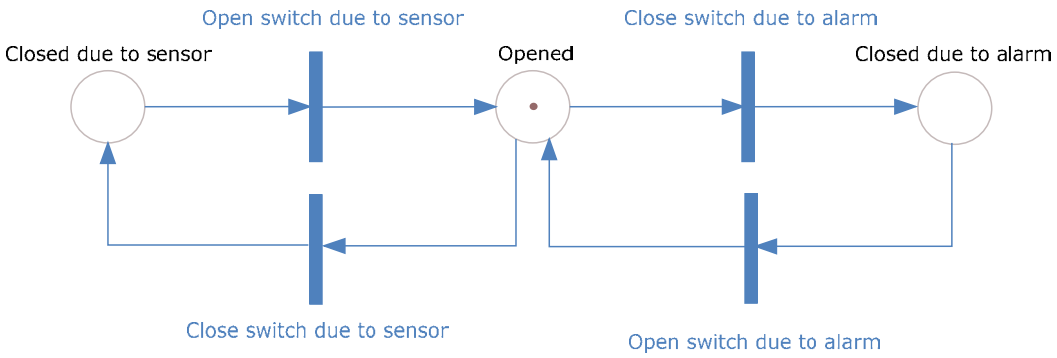


Abbildung 101 Petrinetz, das mit dem DevXML-Code des Gerätes Automated Switch korrespondiert. Ausgehend vom Initialzustand „Switch off“ schließt der Schalter, wenn entweder mittels Webcam ein Dämmerzustand registriert oder zeitabhängig ein Alarm ausgelöst wird.

Die Markierung „Switch off“ wird initial eingenommen. Die Transition „Close switch due to alarm“ ist vom Typ *VariableState* und feuert, wenn die referenzierte Statusvariable *@urn:tu-freiberg.de:serviceID:mediastreamID:RValue* des Gerätes „Camera 1“ den Wert 100 unterschreitet. Hiermit verbunden ist das Schließen des Schalters „Switch 1“ durch Aufruf der Aktion *SetDirectSwitch(true)*. Im Ergebnis wird der Platz „Closed due to sensor“ belegt. Die Transition „Open switch due to sensor“ feuert, wenn der Rotwert 100 überschreitet. Es resultiert die Ursprungsbelegung „Opened“.

Zeitabhängige Alarmereignisse werden auf analoge Weise behandelt. Der Übergang von „Opened“ zu „Closed due to alarm“ vollzieht sich, wenn die Statusvariable *@urn:tu-freiberg.de:serviceID:informationID:AlarmActive* des Gerätes „Alarm 1“ den booleschen Wert *true* annimmt. Dies ist im Alarmzustand der Fall. Mit der Feuerung der zugehörigen Transition „Close switch due to alarm“ wird der Schalter „Switch 1“ durch Aufruf von *SetDirectSwitch(true)* geschlossen. Die Transition „Open switch due to alarm“ bewirkt die Öffnung des Schalters bei Alarmbeendigung durch Aufruf von *SetDirectSwitch(false)*. Die vom Gerät „Alarm 1“ ausgehende Signalisierung erfolgt mit der auf *false* gesetzten Statusvariablen *urn:tu-freiberg.de:serviceID:informationID:AlarmActive*.

6.9 Abschließende Bewertung

Anhand der ausgewählten Entwicklungen wurde nachgewiesen, dass sich die Konzeptvorgaben auf unterschiedlichen Zielplattformen umsetzen lassen. Die experimentelle Validierung umfasste die Neuentwicklung von DevControl-Geräten ebenso wie die Weiterentwicklung bestehender Geräte um DevControl-Funktionalität. Gemeinsam mit der in Kapitel 5 vorgestellten Referenzimplementierung existiert somit ein integrales Framework, das über die avisierten Zielcharakteristika Autokonfiguration, Gerätezuordenbarkeit vor Ort, die Geräteüberwachbarkeit, die Inter-Gerätekommunikation und Automatisierbarkeit von Abläufen verfügt.

Der zweigliedrige Steuerungsansatz stützend auf DevHTML und DevXML hat sich als praxistauglich erwiesen. Mit DevHTML konnten Grundfunktionen der DevControl-Geräte mittels gängiger Browser wie Internet Explorer, Chrome, Mozilla Firefox und Safari genutzt werden. Die ergänzende Eignung von DevXML für die Generierung anspruchsvoller User Interfaces, die geräteübergreifende Kommunikation und Automatisierung wurden ebenfalls demonstriert.

Kapitel 7

Zusammenfassung und Ausblick

Die Motivation zur Befassung mit Konzepten der internetbasierten Gerätesteuerung ist auf die Erkenntnis zurückzuführen, dass die bisher erfolgte technologische Evolution auf diesem Gebiet Anwenderbedürfnisse nur unzureichend einbezieht. In besonderer Weise defizitär ist der Mangel an ganzheitlichen, offenen Frameworks, bei denen die Autokonfiguration, die Gerätezuordenbarkeit vor Ort, die Geräteüberwachbarkeit, die Inter-Gerätekommunikation und die Automatisierbarkeit von Abläufen ausgewogen Berücksichtigung finden.

Proprietäre Einzeltechnologien befördern die weitere Marktfragmentierung und verhindern die Etablierung eines herstellerübergreifenden Lösungsansatzes. Es ist denkbar, dass Hersteller nicht die Zusammenführbarkeit unterschiedlicher Produkte zum Ziel haben, sondern sich in dem Interesse des maximalen Absatzes selbiger erschöpfen. Vor diesem Hintergrund öffnet die vorliegende Arbeit mit einer Bestandsaufnahme von Technologien, die Einzelanforderungen der generischen Gerätesteuerung erfüllen. Sie bilden im weiteren Verlauf das potentielle Architekturfundament. Der Betrachtungsrahmen wird hierbei soweit ausgedehnt, dass relevante Verfahrensschritte vom Geräteanschluss bis zur automatisierten Generierung von User Interfaces abgedeckt sind.

Im Rahmen der Infrastrukturausgestaltung zeigt sich die Vorteilhaftigkeit bewährter Technologien. Ethernet und Wireless LAN entsprechen den Erfordernissen im Hinblick auf Datenübertragungsrate und Quality of Service. Selbst datenintensives Kommunikationsverhalten mit permanentem Informationsfluss, wie z. B. infolge von Audio- und Video-Streaming, führt nachweislich nicht an die Dimensionierungsgrenze. Bezüglich der drahtlosen Übertragung wird im besonderen Maße der Funkstandard 802.11n in Kombination mit dem Verschlüsselungsprotokoll WPA2 dem Anspruch nach hoher Reichweite und Sicherheit gerecht. Industrial Ethernet sowie Feldbussysteme kommen im erarbeiteten Architekturkonzept hingegen nicht zum Tragen. Die Überlegenheit deterministischer gegenüber stochastischen Zugriffsverfahren ist bei den betrachteten Geräteklassen vielmehr theoretischer Natur und – wie gezeigt – nicht von Praxisrelevanz. Hier sind quasi-deterministische Zugriffszeiten von Ethernet und Wireless LAN vollkommen ausreichend. Feldbussysteme sind zudem hochpreisig und erfordern für die Anbindung an das Internet zusätzlich den Einsatz von Gateways. Somit lassen sie sich vorliegend nicht rechtfertigen. Ebenfalls als ungeeignet erweisen sich bei genauerer Betrachtung die WPAN Technologien ZigBee und Bluetooth. Mit maximalen Datenübertragungsraten von 250 kBit/s bzw. 3 MBit/s stellen sie keine Konkurrenz zu WLAN dar.

Eine systematische Analyse der Ad-Hoc-Systeme Jini, Zeroconf sowie UPnP schließt sich an. Die von Sun Microsystems eingeführte Infrastrukturtechnologie Jini gründet auf der Programmiersprache Java. Diese Abhängigkeit schränkt nicht nur im Entwicklungsprozess ein, sondern führt gleichzeitig zu hohem Ressourcenbedarf.

Aufgrund dessen werden programmiersprachenunabhängige Protokolle vorgezogen. Hinsichtlich Zeroconf und UPnP wird technologische Gleichwertigkeit bei konzeptuell analoger Funktionsweise festgestellt. Beide Protokolle unterstützen die dynamische Bildung von Netzwerkverbünden, ohne hierbei Nutzereingriffe zu bedingen. Zahlreiche Beispiele belegen, dass die jeweilige Protokollimplementierung in eigenen Anwendungen mit geringem Aufwand möglich ist. Letztendlich wird UPnP für das Architekturkonzept ausgewählt, da hier auf eine breitere Palette verfügbarer Produkte zurückgegriffen werden kann. Exemplarisch seien hier in Massen vertriebene UPnP-Router, Medienrenderer und -server genannt.

Integraler Bestandteil des Architekturkonzeptes ist überdies die Geräteidentifikation vor Ort auf Basis von Auto-ID-Verfahren wie Optical Character Recognition (OCR) und Radio Frequency Identification (RFID), die eine Alternative zur Listauswahl bieten. Die Funktionsweise beruht auf der Zuweisung von eindeutigen Gerätekennungen zu Identifikationsobjekten wie RFID-Tags.

Im Zusammenhang mit der generischen Gerätebeschreibung stellen sich XML-basierte Sprachen als zweckentsprechend heraus. Sie zeichnen sich durch strukturelle Einfachheit, Plattformunabhängigkeit und Menschenlesbarkeit aus. Darüber hinaus gestatten sie die strikte Trennung von Gerätefunktionalität und User Interface Design. In Anbetracht dieser Sachlage wurde die entwickelte Beschreibungssprache als XML-Dialekt ausgeprägt. Ihre Bezeichnung lautet Device XML (DevXML).

Mit den Sprachelementen von DevXML lassen sich Funktionen und Zustände zueinander in Beziehung setzen und so das dynamische Verhalten von Geräten beschreiben. Die abstrahierte Modellierung ermöglicht unmittelbar die Geräte-Interkommunikation sowie die Automatisierbarkeit von Abläufen. Petrinetze bilden diesbezüglich die theoretische Grundlage. Jeder Gerätezustand entspricht hiernach einer Netzmarkierung, die sich durch Feuern von Transitionen ändern kann. Jeder Transition ist eine Folge von Funktionsaufrufen zuordenbar. Angereichert um Informationen zu transitionsspezifisch erforderlichen Kontrollelementen sind so User Interfaces autogenerierbar. Die formalisierte Herangehensweise hat ferner den Vorteil, dass Aussagen zu Netzeigenschaften algorithmisch abgeleitet werden können. Hierzu zählt die Identifikation ungültiger Netzzustände genauso wie die Ermittlung kürzester Pfade in Verbindung mit geführter Navigation.

Das Device Control (DevControl) Gateway fungiert als zentrale Komponente im Netzwerkverbund. Ausgehend vom DevXML-Code bereitet es User Interfaces auf und stellt diese Steuergeräten zur Verfügung. Zudem gewährleistet es die Zustandssynchronisation zwischen allen beteiligten Kommunikationsteilnehmern. Die zu verwendenden Technologien sind im Übrigen nicht Teil des Konzepts, sondern unterliegen der Implementierungsfreiheit. In der Referenzimplementierung wird Microsoft Silverlight genutzt.

DevControl gliedert sich in zwei Äste. Neben DevXML wird im Architekturkonzept auch Device HTML (DevHTML) spezifiziert mit der Absicht, Geräte mit eingeschränktem Funktionsumfang ebenfalls für Steuerungsaufgaben verwenden zu können. DevHTML bietet bei Weitem nicht den Funktionsumfang von DevXML, insbesondere für die Inter-Gerätekommunikation und die Automatisierbarkeit von Abläufen ist es unzweckmäßig. Dies liegt darin begründet, dass DevHTML darauf beruht, ausschließlich Verweise auf im HTML-Format vorliegende Benutzerschnittstellen zu verwalten. Die Autokonfiguration, die Gerätezuordenbarkeit vor Ort und die Geräteüberwachbarkeit werden sowohl von DevXML als auch DevHTML geboten. Nachfolgende Abbildung zeigt die Zweigliederung von DevControl mitsamt dem Architekturfundament.

Damit die URLs der von einem DevControl-Gateway bereitzustellenden Einstiegsseiten für die DevXML- bzw- DevHTML-basierte Steuerung nicht manuell eingegeben werden müssen, umfasst die Referenzimplementierung ein Browser-Plugin. Dieses ermittelt die Adressen über eine definierte Schnittstelle und ermöglicht deren Aufruf durch Mausklick. Das Plugin ist für den Internet Explorer bestimmt. Auf analoge Weise lassen sich Erweiterungen für sonstige Browser entwickeln.

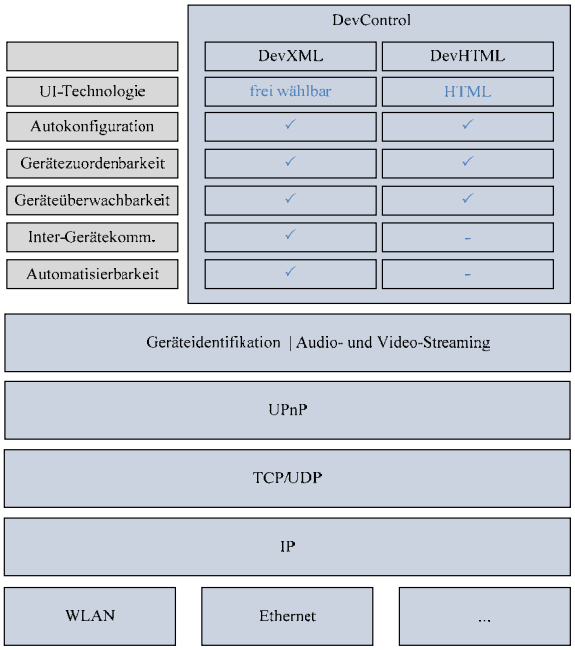


Abbildung 102 Zweigliedriger Ansatz von DevControl. Die Steuerung von Geräten ist mittels DevXML und DevHTML möglich. Beiden Zugriffsmöglichkeiten liegt dasselbe Architekturfundament zugrunde. Sie unterscheiden sich durch die zur Anwendung kommende UI-Technologie sowie die gebotene Funktionalität.

Insbesondere im Umfeld sicherheitskritischer Anwendungen ist die Geräteüberwachbarkeit unabdingbar. Hierfür eignen sich Audio- und Video-Streaming. Die ausgewählten Live-Content-Enkodierer sowie Streaming-Server wurden so in das Gesamtarchitekturkonzept integriert, dass Kompatibilität zu gängigen Wiedergabeprogrammen unterschiedlicher Geräteklassen gegeben ist. Unter Rückgriff auf die Protokolle MMS, WMSP, RTSP/RDP und RTSP/RTP kann Realtime Streaming z. B. mit dem Windows Media Player (WMP), WMP Mobile, Real Player und Safari Mobile durchgeführt werden.

Die Entwicklung ausgewählter DevControl-Geräte verdeutlicht die Vielseitigkeit der vorgestellten Architektur. Der universelle Charakter wird unterstrichen, indem sowohl Weiter- als auch Neuentwicklungen fokussiert werden. Verwendete Programmiersprachen und Geräteklassen variieren. Konkret dienen die Paarungen Standalone Mikrocontroller und Dynamic C, mobiles Endgerät und Java, Windows-Anwendung und C# der Veranschaulichung. Es gelingt dermaßen, DevControl-Konformität bei einem Schalter, einer Heizung, einem Multimeter, einem Telefon, einer Terminalanwendung, einer Jukebox und einer Alarmanwendung herbeizuführen.

Weiterhin wird aufgezeigt, wie mit sogenannten Virtual Devices die Einbettung von Teilfunktionalitäten verschiedenartiger Geräte in einen neuen Kontext erfolgt. Hiermit übertrifft der erzielte Interaktionsgrad den gängiger Systeme. Ein haptisch zu bedienendes Telefon sowie ein zeit- und helligkeitsabhängig gesteuerter Schalter zählen zu den realisierten Geräten. Kontrastiv zu bestehenden Lösungen deckt DevControl zuverlässig alle Schritte von Geräteanschluss bis User Interface-Generierung ab. Es wird dargelegt, dass die Steuerung mit gängigen Browsern wie Internet Explorer, Chrome, Mozilla Firefox und Safari durchführbar ist.

Die vorliegende Arbeit widmet sich in der Hauptsache dem DevControl-Architekturkonzept, das in Bezug auf die Implementierung – wie oben angeführt – technologieoffen ist. Anhand der prototypischen Realisierung in Silverlight wird die grundlegende Wirkweise exemplarisch dargestellt. Stützend auf den hier vorgestellten Modellierungsansatz ist die anwendergerechte Entwicklung von User Interfaces im Rahmen zukünftiger Arbeiten verfolgenswert. Diesbezüglich bietet sich zunächst eine systematische Klassifizierung von Endbenutzern mit deren jeweiligen Bedürfnissen an. Hiervon ausgehend können dann geeignete Technologien selektiert werden. Denkbare Alternativen stellen beispielsweise die Sprachsteuerung mittels VoiceXML, die berührungslose Gestensteuerung mittels Microsoft Kinect und die für mobile Endgeräte optimierte Steuerung mittels Ajax dar. Durch die Einbindung innovativer Bedienkonzepte, die einem rasanten Fortschritt unterliegen, kann die Akzeptanz von DevControl weiter gesteigert werden.

Quellenverzeichnis

/1/ Abts, D.: Masterkurs Client/Server-Programmierung mit Java, Vieweg, 2003.

/2/ Adida, B.: SessionLock: Securing Web Sessions against Eavesdropping, In Proceeding of the 17th international conference on World Wide Web (WWW '08), ACM, New York, NY, USA, S. 517–524, 2008.

/3/ Ailisto, H.; Pohjanheimo, L.; Välkkynen, P.; Strömmer, E.; Tuomisto, T.; Korhonen, I.: Bridging the physical and virtual worlds by local connectivity-based physical selection, 2006.

/4/ Anderson, C.: Pro Business Applications, Apress, 2010.

/5/ Arnold, K.; O'Sullivan, B.; Scheifler, R. W.; Waldo, J.; Wollrath, A.: The Jini Specification, Addison-Wesley Longman, 1999.

/6/ Austerberry, D.: The Technology of Video and Audio Streaming – A robust technical guide to implementing an end-to-end streaming system, Focal Press, 2003.

/7/ Bader, H.; Huber, W.: Jini – Die intelligente Netzwerkarchitektur in Theorie und Praxis, Addison-Wesley, 2000.

/8/ Barr, M.; Massa, A.: Programming Embedded Systems with C and GNU Development Tools, O'Reilly, 2006.

/9/ Biller, S.: Vernetzung von Mikrocontrollern mit dem Ethernet – TCP/IP Kommunikation für Mikrocontroller und Embedded-Systeme, Verlag Dr. Müller, 2008.

/10/ Budde, S.; Barisic, D.: Smart UPnP Transceivers – Design and Implementation of a Prototypical Development Environment for Embedded Network Nodes, Verlag Dr. Müller, 2008.

/11/ Carpenter, T.: Certified Wireless Technology Specialist Study Guide, McGraw-Hill, 2010.

/12/ Case, J.; Mundy, R.; Partain, D.; Stewart, B.: Introduction and Applicability Statements for Internet Standard Management Framework, RFC 3410, 2002. http://www.faqs.org/ftp/rfc/pdf/rfc3410.txt.pdf

/13/ Cheshire, S.; Aboba, B.; Guttmann, E.: Dynamic Configuration of IPv4 Link-Local Addresses, RFC 3927, 2005. http://www.faqs.org/ftp/rfc/pdf/rfc3927.txt.pdf

/14/ Chesire, S.; Krochmal, M.: DNS-Based Service Discovery, Internet Draft, 2010. http://files.dns-sd.org/draft-cheshire-dnsext-dns-sd.txt

/15/ Cheshire, S.; Steinberg, D.: Zero Configuration Networking – The Definitive Guide, O'Reilly, 2006.

/16/ Crane, D.; Pascarello, E.; James, D.: Ajax in action – Das Entwicklerbuch für das Web 2.0, Addison-Wesley, 2006.

/17/ David, R.; Alla, H.: Discrete, Continuous, and Hybrid Petri Nets, Springer, 2005.

/18/ DIN 66008-1: Schrift A für die maschinelle optische Zeichenerkennung – Zeichen und Nennmaße, Juni 1989.

/19/ DIN 66009: Schrift B für die maschinelle optische Zeichenerkennung – Zeichen, Nennmaße und Anordnung auf dem Zeichenträger, September 1977.

/20/ DIN 66225: Schrift H für die maschinelle optische Zeichenerkennung – Zeichen, Schreibregeln und Maße, Januar 1979.

/21/ DIN Einheiten und Begriffe für physikalische Größen – DIN Taschenbuch 22, Beuth Verlag, 1990.

/22/ Droms, R.: Dynamic Host Configuration Protocol, RFC 2131, 1997. http://www.faqs.org/rfcs/rfc2131.html

/23/ Erk, K.; Priese, L.: Theoretische Informatik – Eine umfassende Einführung, Springer, 2008.

/24/ Felser, M.: Ethernet als Feldbus – Kommunikationsmodelle für Industrielle Netzwerke, Infobit 3/2000, S. 21–24, 2000.

/25/ Finkenzeller, K.: RFID-Handbuch – Grundlagen und praktische Anwendungen induktiver Funkanlagen, Transponder und kontaktloser Chipkarten, Carl Hanser Verlag München Wien, 2006.

/26/ Fors, D. S.; Magnusson, B.; Robertz, S. G.; Hedin, G.; Nilsson-Nyman, E.: Ad-hoc composition of pervasive services in the PalCom architecture, In Proceedings of the 2009 international conference on Pervasive services, ACM, New York, NY, USA, S. 83–92, 2009.

/27/ Froitzheim, K.: Multimedia-Kommunikation – Dienste, Protokolle und Technik für Telekommunikation und Computernetze, dpunkt.verlag, 1997.

/28/ Furrer, F.: Industrieautomation mit Ethernet-TCP/IP und Web-Technologie, Hüthig Verlag Heidelberg, 2003.

/29/ Galitz, W.: The Essential Guide to User Interface Design – An introduction to GUI Design Principles and Techniques, Wiley Publishing, 2007.

/30/ Gessler, R.; Krause, T.: Wireless-Netzwerke für den Nahbereich – Eingebettete Funksysteme: Vergleich von standardisierten und proprietären Verfahren, Vieweg + Teubner, 2009.

/31/ Gevatter, H.; Grünhaupt, U.: Handbuch der Mess- und Automatisierungstechnik in der Produktion, Springer, 2006.

/32/ Götze, R.: Dialogmodellierung für multimediale Benutzerschnittstellen, TEUBNER-TEXTE zur Informatik, Teubner, 1995.

/33/ Goland, Y.; Cai, T.; Leach, P.; Gu, Y.; Albright, S.: Simple Service Discovery Protocol/1.0 – Operating without an Arbiter, IETF Internet Draft, 1999. http://tools.ietf.org/html/draft-cai-ssdp-v1-03

/34/ Gulbrandsen, A.; Vixie, P.; Esibov, L.: A DNS RR for specifying the location of services (DNS SRV), RFC 2782, 2000. http://www.faqs.org/ftp/rfc/pdf/rfc2782.txt.pdf

/35/ Guttmann, E.; Perkins, C.; Veizades, J.; Day, M.: Service Location Protocol, Version 2, RFC 2608, 1999. http://www.faqs.org/ftp/rfc/pdf/rfc2608.txt.pdf

/36/ Hammond, J.; Peter, J.: Performance Analysis of Local Computer Networks, Addison-Wesley, 1988.

/37/ Harihar, K; Kurkovsky, S.: Using Jini to enable pervasive computing environments, In Proceedings of the 43rd annual Southeast regional conference, Volume 1, ACM, New York, NY, USA, S. 188–193, 2005.

/38/ Iana: Special-Use IPv4-Addresses, RFC 3330, 2002. http://www.faqs.org/ftp/rfc/pdf/rfc3330.txt.pdf

/39/ Ieee: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats, IEEE 1451.0, 2007.

/40/ Ieee: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Network Capable Application Processor (NCAP) Information Model, IEEE 1451.1, 1999.

/41/ Ieee: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE 1451.2, 1997.

/42/ Ieee: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Digital Communication and Transducer Electronic Data Sheet (TEDS) Formats for Distributed Multidrop Systems, IEEE 1451.3, 2003.

/43/ Ieee: IEEE Standard for A Smart Transducer Interface for Sensors and Actuators – Mixed-Mode Communication Protocols and Transducer Electronic Data Sheet (TEDS) Format, IEEE 1451.4, 2004.

/44/ Ieee: IEEE Standard for a Smart Transducer Interface for Sensors and Actuators – Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats, IEEE 1451.5, 2007.

/45/ Ieee: IEEE Standard for A Smart Transducer Interface for Sensors and Actuators – A High-speed CANopen-based Transducer Network Interface for Intrinsically Safe and Non-intrinsically Safe Applications, IEEE P1451.6, in Arbeit.

/46/ Iso 1073-1: Alphanumeric character sets for optical recognition – Part 1: Character set OCR-A – Shapes and dimensions of the printed image, 1976.

/47/ Iso 1073-2: Alphanumeric character sets for optical recognition – Part 2: Character set OCR-B – Shapes and dimensions of the printed image, 1976.

/48/ Iso/Iec 15420: Information technology – Automatic identification and data capture techniques – Bar code symbology specification – EAN/UPC, 2000.

/49/ Iso/Iec 15909-2: Software and Systems Engineering – High-level Petri Nets, Part 2: Transfer Format, Working Draft Version 0.9.0, 2005.

/50/ Iso/Iec 15438: Information technology – Automatic identification and data capture techniques – PDF417 bar code symbology specification, 2006.

/51/ Jeronimo, M.; Weast, J.: UPnP Design by Example – A Software Developer's Guide to Universal Plug and Play, Intel Press, 2003.

/52/ Jesse, R.; Rosenbaum, O.: Barcode – Theorie, Lexikon, Software, Verlag Technik, 2000.

/53/ Jiang, X.; Ma, M.; Chen, C. (Hrsg.): Mobile Multimedia Processing – Fundamentals, Methods, and Applications, Springer, 2010.

/54/ Josefsson, S.: The Base16, Base32, and Base64 Data Encodings, RFC 4648, 2006. http://www.faqs.org/rfcs/rfc4648.html

/55/ Keller, A.: Datenübertragung im Kabelnetz – DOCSIS über Hybrid-Fibre-Coax, Springer, 2005.

/56/ Kobylinski, R.: Mac OS X Tiger – Netzwerkgrundlagen, Netzwerkanwendungen, Verzeichnisdienste, Springer, 2006.

/57/ Kochan, V.; Lee, K.; Kochan, R.; Sachenko, A.: Approach to improving network capable application processor based on IEEE 1451 standard, Computer Standards and Interfaces, Volume 28, Issue 2, Selected Papers Presented at IDAACS 2003, S. 141–149, Dezember 2005.

/58/ König, R.; Quäck, L.: Petri-Netze in der Steuerungs- und Digitaltechnik, R. Oldenbourg Verlag München Wien, 1988.

/59/ Kriesel, W.; Sokollik, F.; Helm, P.: KNX/EIB für die Gebäudesystemtechnik in Wohn- und Zweckbau, Hüthig Verlag, 2009.

/60/ Kupris, G.; Sikora, A.: ZigBee – Datenfunk mit IEEE 802.15.4 und ZigBee, Franzis, 2007.

/61/ Kurose, J.; Ross, J.: Computernetzwerke – Der Top-Down-Ansatz, Addison-Wesley, 2008.

/62/ Labiod, H.; Afifi, H.; Santis, C.: Wi-Fi, Bluetooth, ZigBee and WiMax, Springer, 2007.

/63/ Lee, J.: Scalable Continuous Media Streaming Systems – Architecture, Design, Analysis and Implementation, Wiley, 2005.

/64/ Lee, K.: Smart Transducer Interface Standards for Condition Monitoring and Control of Machines, Condition Monitoring and Control for Intelligent Manufacturing, S. 347–372, Springer London, 2006.

/65/ Leland, W.; Taqqu, M.; Willinger, W.; Wilson, D.: On the Self-Similar Nature of Ethernet Traffic, IEEE/ACM Transactions on Networking, Volume 2 Issue 1, 1994.

/66/ Lewis, W.: LAN-Switching und Wireless – CCNA Exploration Companion Guide, Addison Wesley, 2009.

/67/ Linetsky, M.: Programming Microsoft DirectShow, Wordware Publishing, 2002.

/68/ Little, J.; Beres, J.; Hinkson, G.; Rader, D.; Croney, J.: Silverlight 3 – Programmer's Reference, Wrox, 2009.

/69/ Lubbers, P.; Albers, B; Salim, F.: Pro HTML5 Programming – Powerful APIs for Richer Internet Application Development, Apress, 2010.

/70/ Martinez, J.; Silva, M.: A Simple and Fast Algorithm to Obtain All Invariants of a Generalized Petri Net, Informatik-Fachberichte 52: Application and Theory of Petri Nets: Selected Papers from the First and Second European Workshop on Application and Theory of Petri Nets, S. 301–310, Springer, 1982.

/71/ Mccloghrie, K.; Rose, M.: Management Information Base for Network Management of TCP/IP-based internets, RFC 1156, 1990. http://www.faqs.org/ftp/rfc/pdf/rfc1156.txt.pdf

/72/ Mccloghrie, K.; Rose, M.: Management Information Base for Network Management of TCP/IP-based internets: MIB-II, RFC 1213, 1991. http://www.faqs.org/ftp/rfc/pdf/rfc1213.txt.pdf

/73/ Merz, H.; Hansemann, T.; Hübner, C.: Gebäudeautomation – Kommunikationssysteme mit EIB/KNX, LON und BACnet, Hanser Verlag, 2010.

/74/ Microsoft Corporation: Microsoft Media Server (MMS) Protocol Specification, November 2009. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-MMSP%5D.pdf

/75/ Microsoft Corporation: Windows Media HTTP Streaming Protocol Specification, November 2009. http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/%5BMS-WMSP%5D.pdf

/76/ Mockapetris, P.: Domain Names – Concepts and Facilities, RFC 1035, 1987. http://www.faqs.org/ftp/rfc/pdf/rfc1034.txt.pdf

/77/ Mockapetris, P.: Domain Names – Implementation and Specification, RFC 1035, 1987. http://www.faqs.org/ftp/rfc/pdf/rfc1035.txt.pdf

/78/ Nichols, J.; Myers, B. A.: Creating a lightweight user interface description language: An overview and analysis of the personal universal controller project, ACM, Transactions on Computer-Human Interaction 16, 4, Article 17, 2009.

/79/ Pantos, R.; May, W.: HTTP Live Streaming, IETF Internet-Draft, 2010. http://tools.ietf.org/pdf/draft-pantos-http-live-streaming-05.pdf

/80/ Plummer, D.: An Ethernet Address Resolution Protocol – or – Converting Network Protocol Addresses to 48 bit Ethernet Address for Transmission on Ethernet Hardware, RFC 826, 1982.. http://www.faqs.org/rfcs/rfc826.html

/81/ Prakash, I.; Warrier, U.: InternetGatewayDevice:1 – Device Template Version 1.01, 2001. http://www.upnp.org/standardizeddcps/documents/upnp_igd_internetgatewaydevice%201.0.pdf

/82/ Priese, L.; Wimmel, H.: Petri-Netze, Springer, 2008.

/83/ Rech, J.: Wireless LANs, 802.11-WLAN-Technologie und praktische Umsetzung im Detail; Heise, 2008.

/84/ Reisig, W.: Petrinetze, Eine Einführung, Springer Verlag, 1990.

/85/ Rekhter, W.; Moskowitz, B.; Karrenberg, D.; Groot, G.; Lear, E.: Address Allocation for Private Internets, RFC 1918, 1996. http://www.faqs.org/rfcs/rfc1918.html

/86/ Rosenberg, J.; Mahy, R.; Matthews, P.; Wing, D.: Session Traversal Utilities for NAT (STUN), RFC 5389, 2008. http://www.faqs.org/rfcs/rfc5389.html

/87/ Rosenberg, J.; Schulzrinne, H.; Camarillo, H.; Johnston, A.; Peterson, J.; Sparks, R.; Handley, M.; Schooler, E.: SIP: Session Initiation Protocol, RFC 3261, 2002. http://www.faqs.org/ftp/rfc/pdf/rfc3261.txt.pdf

/88/ Rupp, S.; Siegmund, G.; Lautenschlager, W.: SIP – Multimediale Dienste im Internet – Grundlagen, Architektur, Anwendungen, dpunkt.verlag, 2002.

/89/ Sales, T.; Sales, L.; Almeida, H.; Perkusich, A.: Enabling user authentication and authorization to support context-aware UPnP applications, In Proceedings of the XV Brazilian Symposium on Multimedia and the Web (WebMedia '09), ACM, New York, NY, USA, Article 3, 8 Seiten, 2009.

/90/ Sauter, M.: Grundkurs Mobile Kommunikationssysteme – Von UMTS und HSDPA, GSM und GPRS zu Wireless LAN und Bluetooth Piconetzen, Vieweg, 2008.

/91/ Schneider, U.; Werner, D.: Taschenbuch der Informatik, Fachbuchverlag Leipzig, 2007.

/92/ Schnell, G.; Wiedemann, B.: Bussysteme in der Automatisierungs- und Prozesstechnik – Grundlagen, Systeme und Trends der industriellen Kommunikation, Vieweg + Teubner, 2008.

/93/ Schulzrinne, H.; Casner, S.; Frederick, R.; Jacobson, V.: RTP – A Transport Protocol for Real-Time Applications, RFC 1889, 1996. http://www.faqs.org/ftp/rfc/pdf/rfc1889.txt.pdf

/94/ Schulzrinne, H.; Rao, A.; Lanphier, R.: Real Time Streaming Protocol (RTSP), RFC 2326, 1998. http://www.faqs.org/ftp/rfc/pdf/rfc2326.txt.pdf

/95/ Schwartz, M.: Telecommunication Networks – Protocols, Modeling and Analysis, Addison-Wesley, 1988.

/96/ Sedgewick, R.: Algorithmen, Addison-Wesley, 2002.

/97/ Shirazi, A. S.; Winkler, C.; Schmidt, A.: SENSE-SATION – An extensible platform for integration of phones into the Web, Internet of Things (IOT), Tokio, Japan, 2010.

/98/ Shirehjini, A. A. N.: A generic UPnP architecture for ambient intelligence meeting rooms and a control point allowing for integrated 2D and 3D interaction, In Proceedings of the 2005 joint conference on Smart objects and ambient intelligence, ACM, New York, NY, USA, S. 207–212, 2005.

/99/ Sikora, A.: ZigBee: Grundlagen und Applikationen – der IEEE-Standard 802.15.4 als Basis, Elektronik Wireless 1/2004, S. 18–24, 2004.

/100/ Song, E.; Lee, K.: Sensor Network based on IEEE 1451.0 and IEEE p1451.2-RS232, Proceedings IIMTC 2008 – IEEE International Instrumentation and Measurement Technology Conference, S. 1728–1733, 2008.

/101/ Srisuresh, P.; Egevang, K.: Traditional IP Network Address Translator (Traditional NAT), RFC 3022, 2001. http://www.faqs.org/ftp/rfc/pdf/rfc3022.txt.pdf

/102/ Sun Microsystems Inc.: Jini Device Architecture Specification, Oktober 2009. http://java.sun.com/products/jini/2.0.2/doc/specs/html/devicearch-spec.html

/103/ Stallings, W.: SNMP, SNMPv2, SNMPv3, and RMON1 and 2, Addison-Wesley, 1999.

/104/ Stockhammer, T.: Dynamic adaptive streaming over HTTP – Standards and Design Principles, In Proceedings of the second annual ACM conference on Multimedia systems (MMSys '11), ACM, New York, NY, USA, S. 133–144, 2011.

/105/ Tanenbaum, A.: Computernetzwerke, Prentice Hall, 2002.

/106/ Tayal, A. P.; Patnaik, L. M.: An address assignment for the automatic configuration of mobile ad hoc networks, Personal Ubiquitous Computing 8, 1, S. 47–54, 2004.

/107/ Topley, K.: J2ME in a Nutshell – A Desktop Quick Reference, O'Reilly, 2002.

/108/ Troncy, R.; Mannens, E.; Pfeiffer, S.; Deursen, D.: Media Fragments URI 1.0 – W3C Working Draft 24 June 2010, W3C, 2010. http://www.w3.org/TR/2010/WD-media-frags-20100624/

/109/ Vilei, A.; Convertino, G.; Crudo, F.: A new UPnP architecture for distributed video voice over IP, In Proceedings of the 5th international conference on Mobile and ubiquitous multimedia (MUM '06), ACM, New York, NY, USA, Article 2, 2006.

/110/ Virnich, M.; Posten, K.: Handbuch der codierten Datenträger, Verlag TÜV Rheinland GmbH, 1992.

/111/ Wagner, R.: Safari and WebKit Development for iPhone OS 3.0, Wrox, 2010.

/112/ Watanabe, M.; Takaya, K.; Seo, A.; Hashimoto, M.; Izumida, T.; Mori, A.: A scheme of service discovery and control on ubiquitous devices, In Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters, ACM, New York, NY, USA, S. 322–323, 2004.

/113/ Wei, J; Zhang, N.; Wang, N.; Lenhert, D.; Neilsen, M.; Mizuno, M.: Use of the 'smart transducer' concept and IEEE 1451 standards in system integration for precision agriculture, Computers and Electronics in Agriculture, Volume 48, Issue 3, S. 245–255, September 2005.

/114/ Winkler, J.: JavaScript und Ajax – Das Praxisbuch für Web-Entwickler, Franzis Verlag, 2009.

/115/ Wolf, K. H.; Froitzheim, K.; Weber, M.: Interactive Video and Remote Control via the World Wide Web, In Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Services, Springer-Verlag, London, UK, S. 91–104, 1996.

/116/ Woods, S.; Bryzek, J.; Chen, S.; Cranmer, J.; El-Kareh, E.; Geipel, M.; Gen-Kuong, F.; Houldsworth, J.; LeComte, N.; Lee, K.; Mattes, M.; Rasmussen, D.: IEEE-P1451.2 Smart Transducer Interface Module, Proceedings Sensors Expo Philadelphia, Helmers Publishing, 1996.

Anhang

A0 TEDS am Beispiel eines Drucksensors

Field types:

U8, U16, U32	are unsigned integers of length 8, 16 and 32 bits respectively.
F32	is a single precision IEEE floating point number
STRING	is an array of character bytes
UNITS	is the representation described in Appendix A

Meta TEDS				
Field #	Description	Field Length (Bytes)	Field type	Field Contents
	Data structure related information			
1	Meta-TEDS Length	4	U32	48
2	IEEE 1451 Standards Family Working Group Number	1	U8	2
3	TEDS Major Version Number	2	U16	2
4	Future Extensions Key	1	U8	0 (NONE)
5	CHANNEL_ZERO Industry Extensions Key	1	U8	0 (NONE)
6	End Users' Application Specific TEDS Key	1	U8	0
7	Number of Implemented Channels	1	U8	1
8	String Language Code	1	U8	0
9	Bytes per Character	1	U8	1
	Timing related information			
10	Worst Case Channel Data Model Length	1	U8	2
11	Worst Case Channel Data Repetitions	2	U16	1
12	Worst Case Channel Update Time	4	F32	2.00E-05
13	Worst Case Channel Write Setup Time	4	F32	0
14	Worst Case Channel Read Setup Time	4	F32	8.00E-05
15	Input/Output Response Time	4	F32	5.00E-04
16	Calibration TEDS Write Time	4	F32	0
17	Worst Case Data Clock Frequency	4	U32	2.00E+05
18	Worst Case Channel Sampling Period	4	F32	2.00E-04
19	Worst Case Unit Warm Up Time	4	F32	1
	Channel grouping related information			
20	Channel Groupings Data Sub-Block Length	2	U16	0
21	Number of Channel Groupings = G	0	U8	-
22	Group Name Length	0	U8	-
23	Group Name (<= 255)	0	STRING	-
24	Group Type	0	U8	-
25	Number of Group Members = N	0	U8	-
26	Member Channel Numbers List = M(N) (<= 255)	0	array of U8	-
	Data integrity information			
27	Checksum for Meta-TEDS	2	U16	62856

	Data structure related information			
28	Meta-Identification TEDS Length	4	U32	310
	Identification related information			
29	Manufacturer's Identification Length	1	U8	55
30	Manufacturer's Identification (<= 255)	55	STRING	Texas Instruments Incorporated Control Product Division
31	Model Number Length	1	U8	9
32	Model Number (<= 255)	9	STRING	EX3514.XX
33	Revision Code Length	1	U8	2
34	Revision Code	2	STRING	01
35	Serial Number Length	1	U8	5
36	Serial Number (<= 255)	5	STRING	SN-01
37	Date Code Length	1	U8	25
38	Date Code (<= 255)	25	STRING	November 1, 1995, Shift 1
39	Product Description Length	2	U16	205
40	Product Description (<= 65535)	205	STRING	Description: Ratiometric Pressure Transducer Part Number: EX3514.XX Serial Number: SN-01 Pressure Range: 0 To 3000 PSIA Input Voltage: 5 Vdc Output Voltage: 0 To 5 Vdc Temperature Range: -40 To 85° C
	Data integrity information data sub-block			
41	Checksum for Meta-Identification TEDS	2	U16	38702

Channel TEDS				
Field #	Description	Field Length (Bytes)	Field type	Field Contents
	Data structure related information			
42	Channel TEDS Length	4	U32	80
43	Calibration Key	1	U8	1 (FIXED)
44	Industry Extension Key	1	U8	0 (NONE)
	Transducer related information			
45	Lower Range Limit	4	F32	0
46	Upper Range Limit	4	F32	20684190
47	Physical Units	10	UNITS	Pa (0,128,128,126,130, 124,128,128,128,128)
48	Unit Type Key	1	U8	0 (SENSOR)
49	Unit Warm Up Time	4	F32	1
50	Self Test Key	1	U8	0 (NONE)
51	Uncertainty	4	F32	206842
	Data converter related information			
52	Channel Data Model	1	U8	0 (N BYTE)
53	Channel Data Model Length	1	U8	2
54	Channel Model Significant Bits	2	U16	12
55	Channel Data Repetitions	2	U16	1
56	Series Increment	4	F32	0
57	Series Units	10	UNITS	0
58	Channel Update Time	4	F32	2.00E-05
59	Channel Write Setup Time	4	F32	0
60	Channel Read Setup Time	4	F32	8.00E-05
61	Data Clock Frequency	4	U32	2.00E+05
62	Channel Sampling Period	4	F32	2.00E-04
63	Timing Correction	4	F32	0
64	Trigger Accuracy	4	F32	5.00E-06
	Data integrity information			
65	Checksum for Channel TEDS	2	U16	59968
	Data structure related information			
66	Channel Identification TEDS Length	4	U32	8
	Identification related information			
67	Manufacturer's Identification Length	1	U8	0
68	Manufacturer's Identification (<= 255)	0	STRING	-
69	Model Number Length	1	U8	0
70	Model Number (<= 255)	0	STRING	-
71	Revision Code Length	1	U8	0
72	Revision Code (<= 255)	0	STRING	-
73	Serial Number Length	1	U8	0
74	Serial Number (<= 255)	0	STRING	-
75	Channel Description Length	2	U16	0
76	Channel Description (<= 65535)	0	STRING	-
	Data Integrity information			
77	Checksum for Channel Identification TEDS	2	U16	65527

Calibration TEDS				
Field #	Description	Field Length (Bytes)	Field type	Field Contents
	Data structure related information			
78	Calibration TEDS Length	4	U32	99
	Calibration related information			
79	Last Calibration Date-Time	4	U32	0
80	Calibration Interval	4	U32	0
81	Number of Correction Input Channels = n	1	U8	1
82	Correction Input Channel List	1	U8	1
83	Correction Input Channel-Key List	1	U8	0
84	Channel Degree List = D(k)	1	U8	1
85	Number of Segments List = N_k	1	U8	5
86	Segment Boundary Values Table (Pa)	24	F32	0
	(segment 1 high boundary)		F32	4136838
	(segment 2 high boundary)		F32	8273676
	(segment 3 high boundary)		F32	12410514
	(segment 4 high boundary)		F32	16547352
	(segment 5 high boundary)		F32	20684190
87	Segment Offset Values Table (Pa)	20		
	(segment 1 offset)		F32	5051
	(segment 2 offset)		F32	5051
	(segment 3 offset)		F32	5051
	(segment 4 offset)		F32	5051
	(segment 5 offset)		F32	5051
88	Multinomial Coefficients	40		
	A_{00} (Pa)		F32	-126372
	A_{01} (Pa/count)		F32	5244
	A_{10}		F32	-44141
	A_{11}		F32	5144
	A_{20}		F32	111220
	A_{21}		F32	5049
	A_{30}		F32	331826
	A_{31}		F32	4959
	A_{40}		F32	610811
	A_{41}		F32	4874
	Data integrity information			
89	Checksum for Calibration TEDS	2	U16	57092

A1 DevXML Schemadefinition

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2009 (http://www.altova.com) by Schneider (none) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
   <xs:element name="devxml">
      <xs:complexType>
         <xs:sequence>
            <xs:element name="places">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="place" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="statevariable" minOccurs="0" maxOccurs="unbounded">
                                 <xs:complexType>
                                    <xs:attribute name="name" use="required"/>
                                    <xs:attribute name="title" use="required"/>
                                    <xs:attribute name="unit" use="required"/>
                                    <xs:attribute name="deviceUDN" use="optional"/>
                                    <xs:attribute name="deviceAddName"/>
                                 </xs:complexType>
                              </xs:element>
                              <xs:element name="camera" minOccurs="0" maxOccurs="unbounded">
                                 <xs:complexType>
                                    <xs:attribute name="name" use="required"/>
                                    <xs:attribute name="width" use="required"/>
                                    <xs:attribute name="height" use="required"/>
                                 </xs:complexType>
                              </xs:element>
                           </xs:sequence>
                           <xs:attribute name="name" use="required"/>
                           <xs:attribute name="x" use="required"/>
                           <xs:attribute name="y" use="required"/>
                           <xs:attribute name="isActive" use="required"/>
                        </xs:complexType>
                     </xs:element>
                  </xs:sequence>
               </xs:complexType>
            </xs:element>
            <xs:element name="transitions" minOccurs="0">
               <xs:complexType>
                  <xs:sequence>
                     <xs:element name="transition" maxOccurs="unbounded">
                        <xs:complexType>
                           <xs:sequence>
                              <xs:element name="transitioncontrol" minOccurs="0">
                                 <xs:complexType>
                                    <xs:sequence>
                                       <xs:element name="upnpcall" minOccurs="0" maxOccurs="unbounded">
                                          <xs:complexType>
                                             <xs:sequence>
                                                <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
                                                   <xs:complexType>
                                                      <xs:simpleContent>
                                                         <xs:extension base="xs:string">
                                                            <xs:attribute name="name" use="required"/>
                                                            <xs:attribute name="type" use="required"/>
                                                            <xs:attribute name="dir" use="required"/>
                                                            <xs:attribute name="deviceUDN"/>
                                                            <xs:attribute name="deviceAddName"/>
                                                         </xs:extension>
                                                      </xs:simpleContent>
                                                   </xs:complexType>
                                                </xs:element>
                                                <xs:element name="return" minOccurs="0"/>
                                             </xs:sequence>
```

```
67                                  <xs:attribute name="service" use="required"/>
68                                  <xs:attribute name="action" use="required"/>
69                                  <xs:attribute name="deviceUDN"/>
70                                  <xs:attribute name="deviceAddName"/>
71                               </xs:complexType>
72                            </xs:element>
73                            <xs:element name="controlelement" minOccurs="0" maxOccurs="unbounded">
74                               <xs:complexType>
75                                  <xs:sequence>
76                                     <xs:element name="select" minOccurs="0" maxOccurs="unbounded">
77                                        <xs:complexType>
78                                           <xs:attribute name="name" use="required"/>
79                                           <xs:attribute name="show" use="required"/>
80                                           <xs:attribute name="output" use="optional"/>
81                                        </xs:complexType>
82                                     </xs:element>
83                                  </xs:sequence>
84                                  <xs:attribute name="name" use="required"/>
85                                  <xs:attribute name="type" use="required"/>
86                                  <xs:attribute name="input" use="required"/>
87                                  <xs:attribute name="title" use="optional"/>
88                                  <xs:attribute name="output" use="optional"/>
89                                  <xs:attribute name="min" use="optional"/>
90                                  <xs:attribute name="max" use="optional"/>
91                                  <xs:attribute name="deviceUDN"/>
92                                  <xs:attribute name="deviceAddName"/>
93                               </xs:complexType>
94                            </xs:element>
95                         </xs:sequence>
96                      </xs:complexType>
97                   </xs:element>
98                   <xs:element name="transitionaction">
99                      <xs:complexType>
100                        <xs:sequence>
101                           <xs:element name="firecondition" maxOccurs="unbounded">
102                              <xs:complexType>
103                                 <xs:attribute name="type" use="required"/>
104                                 <xs:attribute name="subtype" use="optional"/>
105                                 <xs:attribute name="UIname" use="optional"/>
106                                 <xs:attribute name="varname" use="optional"/>
107                                 <xs:attribute name="varvalue" use="optional"/>
108                                 <xs:attribute name="varoperator" use="optional"/>
109                                 <xs:attribute name="vartype" use="optional"/>
110                                 <xs:attribute name="deviceUDN"/>
111                                 <xs:attribute name="deviceAddName"/>
112                              </xs:complexType>
113                           </xs:element>
114                           <xs:element name="upnpcall" minOccurs="0" maxOccurs="unbounded">
115                              <xs:complexType>
116                                 <xs:sequence>
117                                    <xs:element name="param" minOccurs="0" maxOccurs="unbounded">
118                                       <xs:complexType>
119                                          <xs:simpleContent>
120                                             <xs:extension base="xs:string">
121                                                <xs:attribute name="name" use="required"/>
122                                                <xs:attribute name="type" use="required"/>
123                                                <xs:attribute name="dir" use="required"/>
124                                                <xs:attribute name="deviceUDN"/>
125                                                <xs:attribute name="deviceAddName"/>
126                                             </xs:extension>
127                                          </xs:simpleContent>
128                                       </xs:complexType>
129                                    </xs:element>
130                                    <xs:element name="return" minOccurs="0"/>
131                                 </xs:sequence>
132                                 <xs:attribute name="service" use="required"/>
```

```
                                        <xs:attribute name="action" use="required"/>
                                        <xs:attribute name="deviceUDN"/>
                                        <xs:attribute name="deviceAddName"/>
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                    </xs:element>
                </xs:sequence>
                <xs:attribute name="name" use="required"/>
                <xs:attribute name="id" use="required"/>
                <xs:attribute name="externalUI" type="xs:boolean" use="optional"/>
                <xs:attribute name="autoFireable"/>
                <xs:attribute name="targetTransition"/>
                <xs:attribute name="descriptionUrl"/>
            </xs:complexType>
        </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="preedges" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="preedge" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="placex" use="required"/>
          <xs:attribute name="placey" use="required"/>
          <xs:attribute name="transitionid" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="postedges" minOccurs="0">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="postedge" maxOccurs="unbounded">
        <xs:complexType>
          <xs:attribute name="placex" use="required"/>
          <xs:attribute name="placey" use="required"/>
          <xs:attribute name="transitionid" use="required"/>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

A2 DevXML-Code Switch

```
<devxml>
  <places>
    <place name='Closed' x='0' y='0' isActive='false'></place>
    <place name='Opened' x='1' y='0' isActive='true'></place>
  </places>
  <transitions>
    <transition name='Open Switch' id='1'>
      <transitioncontrol>
        <controlelement input='Open Switch' type='button' name='myButton'/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type='UIEvent' subtype='OnClick' UIname='myButton'/>
        <!-- UPnP-Aktion SetDirectSwitch aufrufen   -->
        <upnpcall service='urn:tu-freiberg.de:serviceID:switchItID:1' action='SetDirectSwitch'>
          <param name='switch_state' type='boolean' dir='in'>
            false
          </param>
        </upnpcall>
      </transitionaction>
    </transition>
    <transition name='Close Switch' id='2'>
      <transitioncontrol>
        <controlelement input='Close Switch' type='button' name='myButton'/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type='UIEvent' subtype='OnClick' UIname='myButton'/>
        <!-- UPnP-Aktion SetDirectSwitch aufrufen   -->
        <upnpcall service='urn:tu-freiberg.de:serviceID:switchItID:1' action='SetDirectSwitch'>
          <param name='switch_state' type='boolean' dir='in'>
            true
          </param>
        </upnpcall>
      </transitionaction>
    </transition>
  </transitions>
  <preedges>
    <preedge placex='0' placey='0' transitionid='1'></preedge>
    <preedge placex='1' placey='0' transitionid='2'></preedge>
  </preedges>
  <postedges>
    <postedge placex='1' placey='0' transitionid='1'></postedge>
    <postedge placex='0' placey='0' transitionid='2'></postedge>
  </postedges>
</devxml>
```

A3 DevXML-Code Heating

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
 <places>
  <place name="Manual Setting" x="0" y="0" isActive="false">
   <statevariable name="@urn:tu-freiberg.de:serviceID:heatingServiceID:HeatingWaterValue"
   title="Heating Water Temperature" unit="[24]=129"/>
   <statevariable name="@urn:tu-freiberg.de:serviceID:heatingServiceID:DomesticWaterValue"
   title="Domestic Water Temperature" unit="[24]=129"/>
   <camera name="Camera1" width="100" height="100"/>
  </place>
  <place name="Remote Setting" x="1" y="0" isActive="true">
   <statevariable name="@urn:tu-freiberg.de:serviceID:heatingServiceID:HeatingWaterValue"
   title="Heating Water Temperature" unit="[24]=129"/>
   <statevariable name="@urn:tu-freiberg.de:serviceID:heatingServiceID:DomesticWaterValue"
   title="Domestic Water Temperature" unit="[24]=129"/>
   <camera name="Camera1" width="100" height="100"/>
  </place>
 </places>
 <transitions>
  <transition name="Switch to remote setting" id="1">
   <transitioncontrol>
    <controlelement input="Switch to remote setting" type="button" name="myButton">
      </controlelement>
   </transitioncontrol>
   <transitionaction>
    <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
    <!-- UPnP-Aktion SwitchToRemote aufrufen  -->
    <upnpcall service="urn:tu-freiberg.de:serviceID:heatingServiceID" action="SwitchToRemote">
   </upnpcall>
   </transitionaction>
  </transition>
  <transition name="Switch to manual setting" id="2">
   <transitioncontrol>
    <controlelement input="Switch to manual setting" type="button" name="myButton">
      </controlelement>
   </transitioncontrol>
   <transitionaction>
    <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
    <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:heatingServiceID:DomesticWaterValue'
    varvalue='35' varoperator='=' vartype='string'></firecondition>
    <!-- UPnP-Aktion SwitchToManual aufrufen  -->
    <upnpcall service="urn:tu-freiberg.de:serviceID:heatingServiceID" action="SwitchToManual">
   </upnpcall>
   </transitionaction>
  </transition>
  <transition name="Edit values" id="3">
   <transitioncontrol>
    <controlelement input="@urn:tu-freiberg.de:serviceID:heatingServiceID:HeatingWaterValue"
    title="Heating water" type="slider" min="30" max="75" name="myHeatingSlider" output="@heatingValue">
      </controlelement>
    <controlelement input="@urn:tu-freiberg.de:serviceID:heatingServiceID:DomesticWaterValue"
    title="Domestic water" type="slider" min="35" max="65" name="myDomesticSlider" output="@domesticValue">
      </controlelement>
   </transitioncontrol>
   <transitionaction>
    <firecondition type="UIEvent" subtype="OnValueChanged" UIname="myHeatingSlider"/>
    <firecondition type="UIEvent" subtype="OnValueChanged" UIname="myDomesticSlider"/>
    <!-- UPnP-Aktion EditValues aufrufen  -->
    <upnpcall service="urn:tu-freiberg.de:serviceID:heatingServiceID" action="SetHeatingWaterValue">
     <param name="HeatingWaterValue" type="i2" dir="in">
         @heatingValue
    </param>
    </upnpcall>
    <upnpcall service="urn:tu-freiberg.de:serviceID:heatingServiceID" action="SetDomesticWaterValue">
     <param name="DomesticWaterValue" type="i2" dir="in">
         @domesticValue
```

```
67          </param>
68          </upnpcall>
69        </transitionaction>
70      </transition>
71    </transitions>
72    <preedges>
73      <preedge placex="0" placey="0" transitionid="1"/>
74      <preedge placex="1" placey="0" transitionid="2"/>
75      <preedge placex="1" placey="0" transitionid="3"/>
76    </preedges>
77    <postedges>
78      <postedge placex="1" placey="0" transitionid="1"/>
79      <postedge placex="0" placey="0" transitionid="2"/>
80      <postedge placex="1" placey="0" transitionid="3"/>
81    </postedges>
82  </devxml>
```

A4 DevXML-Code Multimeter

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
  <places>
    <place name='Measurement' x='0' y='0' isActive='true'>
      <statevariable name='@urn:tu-freiberg.de:serviceID:measurementID:MeasuredValue'
      unit='@urn:tu-freiberg.de:serviceID:measurementID:CurrentUnit' title='Measured Value'/>
    </place>
  </places>
</devxml>
```

A5 DevXML-Code Phone

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
   <places>
      <place name="Wait for call" x="0" y="0" isActive="true"/>
   </places>
   <transitions>
      <transition name="Dial number" id="1">
         <transitioncontrol>
            <controlelement type="textbox" input="" output="@number" name="myNumberBox"/>
            <controlelement input="Dial number" type="button" name="myButton">
       </controlelement>
         </transitioncontrol>
         <transitionaction>
            <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
            <upnpcall service="urn:tu-freiberg.de:serviceID:operatorID" action="DialPhoneNumber">
               <param name="PhoneNumber" type="string" dir="in">@number</param>
               <return>@success</return>
            </upnpcall>
         </transitionaction>
      </transition>
   </transitions>
   <preedges>
      <preedge placex="0" placey="0" transitionid="1"/>
   </preedges>
   <postedges>
      <postedge placex="0" placey="0" transitionid="1"/>
   </postedges>
</devxml>
```

A6 DevXML-Code Jukebox

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
  <places>
    <place name="Wait for renderer" x="0" y="0" isActive="true">
    </place>
    <place name="Renderer prepared" x="1" y="0" isActive="false"/>
    <place name="Playing" x="3" y="1" isActive="false">
      <statevariable name="@urn:tu-freiberg.de:serviceID:musicID:Volume" title="Volume" unit="[1]=0"/>
      <statevariable name="@urn:tu-freiberg.de:serviceID:musicID:ActualDate" title="Actual time" unit="[1]=0"/>
    </place>
    <place name="Wait for server" x="0" y="2" isActive="true"/>
    <place name="Wait for title" x="1" y="2" isActive="false"/>
    <place name="Server prepared" x="2" y="2" isActive="false"/>
  </places>
  <transitions>
    <transition name="Choose renderer" id="1" externalUI="true">
      <transitioncontrol>
        <!-- UPnP-Aktion GetRendererList aufrufen  -->
        <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="GetRendererList">
          <param name="XMLResult" type="string" dir="out">@result1</param>
        </upnpcall>
        <!-- Daten auswerten    -->
        <controlelement type="table" name="myRenderer" input="@result1">
          <select name="udn" output="@result2" show="false"/>
          <select name="friendlyName" output="@result3" show="true"/>
        </controlelement>
        <controlelement input="@result3" type="button" name="myButton"/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
        <!--
    <firecondition type="UIEvent" subtype="OnSelectionChanged" UIname="myRenderer"/>
        -->
        <!-- UPnP-Aktion SetRenderer aufrufen  -->
        <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="SetRenderer">
          <param name="URNParam" type="string" dir="in">@result2</param>
        </upnpcall>
      </transitionaction>
    </transition>
    <transition name="Change renderer" id="2" externalUI="true" targetTransition="false" autoFireable="true">
      <transitioncontrol>
        <controlelement input="Change renderer" type="button" name="myButton"/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
      </transitionaction>
    </transition>
    <transition name="Start playing" id="3" externalUI="true">
      <transitioncontrol>
        <controlelement input="Start playing" type="button" name="myButton"/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
        <!-- UPnP-Aktion start aufrufen  -->
        <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="Start">
        </upnpcall>
      </transitionaction>
    </transition>
    <transition name="Stop playing" id="4" externalUI="true">
      <transitioncontrol>
        <controlelement input="Stop playing" type="button" name="myButton"/>
      </transitioncontrol>
      <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
        <firecondition type="VariableState" varname="@urn:tu-freiberg.de:serviceID:musicID:Playing"
        varvalue="false" varoperator="=" vartype="boolean"/>
```

```
67        <!-- UPnP-Aktion start aufrufen  -->
68        <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="Stop">
69        </upnpcall>
70      </transitionaction>
71    </transition>
72    <transition name="Change title" id="5" externalUI="true" targetTransition="false" autoFireable="true">
73      <transitioncontrol>
74        <controlelement input="Change title" type="button" name="myButton"/>
75      </transitioncontrol>
76      <transitionaction>
77        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
78      </transitionaction>
79    </transition>
80    <transition name="Choose title" id="6" externalUI="true">
81      <transitioncontrol>
82        <!-- UPnP-Aktion GetMusicContentList aufrufen  -->
83        <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="GetMusicContentList">
84          <param name="XMLResult" type="string" dir="out">
85           @result1
86          </param>
87        </upnpcall>
88        <!-- Daten auswerten    -->
89        <controlelement input="@result1" type="table" name="myTitle">
90          <select name="uri" output="@result2" show="false"/>
91          <select name="title" show="true" output="@result3"/>
92          <select name="author" show="true"/>
93        </controlelement>
94        <controlelement input="@result3" type="button" name="myButton">
95        </controlelement>
96      </transitioncontrol>
97      <transitionaction>
98        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
99        <!--
100       <firecondition type="VariableState" varname="@result3" varvalue="Amanda" varoperator="!=" vartype="string"/>
101       -->
102       <!-- UPnP-Aktion SetTitle aufrufen  -->
103       <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="SetTitle">
104         <param name="URI" type="string" dir="in">
105          @result2
106         </param>
107       </upnpcall>
108     </transitionaction>
109   </transition>
110   <transition name="Change server" id="7" externalUI="true" targetTransition="false" autoFireable="true">
111     <transitioncontrol>
112       <controlelement input="Change server" type="button" name="myButton">
113         </controlelement>
114     </transitioncontrol>
115     <transitionaction>
116       <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
117     </transitionaction>
118   </transition>
119   <transition name="Choose server" id="8" externalUI="true">
120     <transitioncontrol>
121       <!-- UPnP-Aktion GetServerList aufrufen  -->
122       <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="GetServerList">
123         <param name="XMLResult" type="string" dir="out">
124         @result1
125         </param>
126       </upnpcall>
127       <!-- Daten auswerten    -->
128       <controlelement input="@result1" type="table" name="myServer">
129         <select name="udn" output="@result2" show="false"/>
130         <select name="friendlyName" output="@result3" show="true"/>
131       </controlelement>
132       <controlelement input="@result3" type="button" name="myButton">
```

```
133         </controlelement>
134       </transitioncontrol>
135       <transitionaction>
136         <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
137         <!-- UPnP-Aktion SetServer aufrufen  -->
138         <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="SetServer">
139           <param name="URNParam" type="string" dir="in">
140           @result2
141           </param>
142         </upnpcall>
143       </transitionaction>
144     </transition>
145     <transition name="Modify volume" id="9" externalUI="true">
146       <transitioncontrol>
147     <controlelement input="@urn:tu-freiberg.de:serviceID:musicID:Volume" title="Volume" type="slider" min="0" max="100"
148     name="myVolumeSlider" output="@volumeValue">
149     </controlelement>
150   </transitioncontrol>
151       <transitionaction>
152         <firecondition type="UIEvent" subtype="OnValueChanged" UIname="myVolumeSlider"/>
153         <!-- UPnP-Aktion start aufrufen  -->
154         <upnpcall service="urn:tu-freiberg.de:serviceID:musicID" action="SetVolume">
155           <param name="Volume" type="ui1" dir="in">
156     @volumeValue
157       </param>
158         </upnpcall>
159       </transitionaction>
160     </transition>
161   </transitions>
162   <preedges>
163     <preedge placex="0" placey="0" transitionid="1"/>
164     <preedge placex="1" placey="0" transitionid="2"/>
165     <preedge placex="1" placey="0" transitionid="3"/>
166     <preedge placex="2" placey="2" transitionid="3"/>
167     <preedge placex="3" placey="1" transitionid="4"/>
168     <preedge placex="2" placey="2" transitionid="5"/>
169     <preedge placex="1" placey="2" transitionid="6"/>
170     <preedge placex="1" placey="2" transitionid="7"/>
171     <preedge placex="0" placey="2" transitionid="8"/>
172     <preedge placex="3" placey="1" transitionid="9"/>
173   </preedges>
174   <postedges>
175     <postedge placex="1" placey="0" transitionid="1"/>
176     <postedge placex="0" placey="0" transitionid="2"/>
177     <postedge placex="3" placey="1" transitionid="3"/>
178     <postedge placex="2" placey="2" transitionid="4"/>
179     <postedge placex="1" placey="0" transitionid="4"/>
180     <postedge placex="1" placey="2" transitionid="5"/>
181     <postedge placex="2" placey="2" transitionid="6"/>
182     <postedge placex="0" placey="2" transitionid="7"/>
183     <postedge placex="1" placey="2" transitionid="8"/>
184     <postedge placex="3" placey="1" transitionid="9"/>
185   </postedges>
186 </devxml>
```

A7 DevXML-Code Alarm

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
  <places>
    <place name='Alarm on' x='0' y='0' isActive='false'></place>
    <place name='Alarm off' x='1' y='0' isActive='true'></place>
  </places>
  <transitions>
        <transition id='1' name='Set alarm on'>
          <transitionaction>
            <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:informationID:AlarmActive'
            varvalue='true' varoperator='=' vartype='boolean'></firecondition>
          </transitionaction>
        </transition>
        <transition id='2' name='Set alarm off'>
          <transitionaction>
            <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:informationID:AlarmActive'
            varvalue='false' varoperator='=' vartype='boolean'></firecondition>
          </transitionaction>
        </transition>
    </transitions>
    <preedges>
      <preedge transitionid='1' placex='1' placey='0'></preedge>
      <preedge transitionid='2' placex='0' placey='0'></preedge>
    </preedges>
    <postedges>
      <postedge transitionid='1' placex='0' placey='0'></postedge>
      <postedge transitionid='2' placex='1' placey='0'></postedge>
    </postedges>
</devxml>
```

A8 DevXML-Code Haptic Phone

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
  <places>
    <place name='Wait for reading' x='0' y='1' isActive='true'>
    <statevariable name='@urn:tu-freiberg.de:serviceID:devXMLID:StringVar' unit='[1]=0' title='Phone number'/>
    </place>
    <place name='Data read' x='1' y='0' isActive='false'>
      <statevariable name='@urn:tu-freiberg.de:serviceID:devXMLID:StringVar' unit='[1]=0' title='Phone number'/>
    </place>
   <place name='Wait for call' x='2' y='1' isActive='false'>
      <statevariable name='@urn:tu-freiberg.de:serviceID:devXMLID:StringVar' unit='[1]=0' title='Phone number'/>
    </place>
  </places>
  <transitions>
    <transition name='Read data' id='1'>
        <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
          <firecondition type="always"/>
          <upnpcall deviceAddName='RFID1' service="urn:tu-freiberg.de:serviceID:autoidID" action="GetData">
        <return>@number</return>
          </upnpcall>
        <upnpcall service="urn:tu-freiberg.de:serviceID:devXMLID" action="SetStringVar">
          <param name='StringVar' type='string' dir='in'>@number</param>
          </upnpcall>
        </transitionaction>
    </transition>
   <transition name='Back to start' id='2'>
      <transitionaction>
        <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:devXMLID:StringVar' varoperator='='
        varvalue='' vartype='string'/>
      </transitionaction>
    </transition>
    <transition name='Dial number' id='3'>
      <transitionaction>
        <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:devXMLID:StringVar' varoperator='!='
        varvalue='' vartype='string'/>
        <upnpcall deviceAddName='Phone1' service='urn:tu-freiberg.de:serviceID:operatorID' action='DialPhoneNumber'>
        <param name="PhoneNumber" type="string" dir="in">@urn:tu-freiberg.de:serviceID:devXMLID:StringVar</param>
        </upnpcall>
      </transitionaction>
    </transition>
    <transition name='Back to start' id='4'>
        <transitioncontrol>
          <controlelement input="Back to start" type="button" name="myButton">
          </controlelement>
        </transitioncontrol>
      <transitionaction>
        <firecondition type="UIEvent" subtype="OnClick" UIname="myButton"/>
      </transitionaction>
    </transition>
  </transitions>
  <preedges>
    <preedge placex="0" placey="1" transitionid="1"/>
    <preedge placex="1" placey="0" transitionid="2"/>
    <preedge placex="1" placey="0" transitionid="3"/>
    <preedge placex="2" placey="1" transitionid="4"/>
  </preedges>
  <postedges>
    <postedge placex="1" placey="0" transitionid="1"/>
    <postedge placex="0" placey="1" transitionid="2"/>
    <postedge placex="2" placey="1" transitionid="3"/>
    <postedge placex="0" placey="1" transitionid="4"/>
  </postedges>
</devxml>
```

A9 DevXML-Code Automated Switch

```
<devxml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="D:\DOKUME~1\Hoffmann\Desktop\Diss(01-10-2009)\devXMLSchema.xsd">
 <places>
  <place name='Closed due to sensor' x='0' y='0' isActive='false'>
   <statevariable deviceAddName='Camera1'
   name='@urn:tu-freiberg.de:serviceID:mediastreamID:RValue' unit='[1]=0' title='Red Value'/>
  </place>
  <place name='Opened' x='1' y='0' isActive='true'>
   <statevariable deviceAddName='Camera1'
   name='@urn:tu-freiberg.de:serviceID:mediastreamID:RValue' unit='[1]=0' title='Red Value'/>
  </place>
  <place name='Closed due to alarm' x='2' y='0' isActive='false'></place>
 </places>
 <transitions>
  <transition name='Open switch due to sensor' id='1'>
   <transitionaction>
    <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:mediastreamID:RValue'
    deviceAddName='Camera1' varoperator='&gt;' varvalue='100' vartype='ui1'/>
    <upnpcall deviceAddName='Switch1' service="urn:tu-freiberg.de:serviceID:switchItID:1"
    action="SetDirectSwitch">
     <param name='switch_state' type='boolean' dir='in'>
     false
     </param>
    </upnpcall>
   </transitionaction>
  </transition>
  <transition name='Close switch due to sensor' id='2'>
   <transitionaction>
    <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:mediastreamID:RValue'
    deviceAddName='Camera1' varoperator='&lt;' varvalue='100' vartype='ui1'/>
    <upnpcall deviceAddName='Switch1' service='urn:tu-freiberg.de:serviceID:switchItID:1'
    action='SetDirectSwitch'>
     <param name="switch_state" type="boolean" dir="in">
     true
     </param>
    </upnpcall>
   </transitionaction>
  </transition>
  <transition name='Close switch due to alarm' id='3'>
   <transitionaction>
    <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:informationID:AlarmActive'
    deviceAddName='Alarm1' varoperator='=' varvalue='true' vartype='boolean'/>
    <upnpcall deviceAddName='Switch1' service='urn:tu-freiberg.de:serviceID:switchItID:1'
    action='SetDirectSwitch'>
     <param name="switch_state" type="boolean" dir="in">
     true
     </param>
    </upnpcall>
   </transitionaction>
  </transition>
  <transition name='Open switch due to alarm' id='4'>
   <transitionaction>
    <firecondition type='VariableState' varname='@urn:tu-freiberg.de:serviceID:informationID:AlarmActive'
    deviceAddName='Alarm1' varoperator='=' varvalue='false' vartype='boolean'/>
     <upnpcall deviceAddName='Switch1' service='urn:tu-freiberg.de:serviceID:switchItID:1'
    action='SetDirectSwitch'>
     <param name="switch_state" type="boolean" dir="in">
     false
     </param>
    </upnpcall>
   </transitionaction>
  </transition>
 </transitions>
 <preedges>
  <preedge placex="0" placey="0" transitionid="1"/>
  <preedge placex="1" placey="0" transitionid="2"/>
```

```
67      <preedge placex="1" placey="0" transitionid="3"/>
68      <preedge placex="2" placey="0" transitionid="4"/>
69    </preedges>
70    <postedges>
71      <postedge placex="0" placey="0" transitionid="2"/>
72      <postedge placex="1" placey="0" transitionid="1"/>
73      <postedge placex="1" placey="0" transitionid="4"/>
74      <postedge placex="2" placey="0" transitionid="3"/>
75    </postedges>
76  </devxml>
```