RapidCoop - Robuste Architektur durch geeignete Paradigmen für Kooperativ Interagierende Automobile

Jens Dankert[1], Christian Dernehl[2], Evgeny Kusmenko[3], Univ.-Prof. Dr-Ing. Lutz Eckstein[1], Prof. Dr.-Ing. Stefan Kowalewski[2] und Prof. Dr. Bernhard Rumpe[3]

RWTH Aachen University

Abstract

Der rasche Fortschritt im Bereich des automatisierten Fahrens ermöglicht auch automatisiertes kooperatives Fahrverhalten. Um dieses Verhalten sicher, effizient und nachvollziehbar zu gestalten, müssen entsprechende Architekturen und Modelle zugrunde gelegt werden. Dieser Artikel stellt beschreibt einen Ansatz zur Modellierung und Bildung lokaler Gruppen von Verkehrsteilnehmern, die Local Traffic Systems (LTS). Die Entwicklung solcher komplexer verteilter autonomer Systeme erfordert Sprachen und Modellierungstechniken, welche auf die Domäne zugeschnitten sind. Wir stellen einen effizienten Komponenten- und Konnektor-basierten Modellierungsansatz für LTS unter Verwendung der erweiterbaren Architekturbeschreibungssprache MontiArc vor. Es muss zudem sichergestellt werden, dass das System nicht unsicherer als der aktuelle Stand der Technik ist. Mithilfe der Onlineverifikation können Fehler im System frühzeitig erkannt und die notwendigen Eingriffe vorgenommen werden. Die Ergebnisse der Onlineverifikation werden schließlich den anderen Komponenten zur Verfügung gestellt um so die Güte des Gesamtsystems kontinuierlich zu verbessern.

1 Einleitung

Zeichnete sich vor einigen Jahren der Trend ab, hochautomatisierte Fahrzeuge in Forschungsprojekten zu entwickeln, existieren heute bei jedem größeren Fahrzeughersteller Pläne, derartige Fahrzeuge auch mittelfristig zur Serienreife zu bringen. Moderne Sensorik und Kommunikations-

[1]dankert@ika.rwth-aachen.de, Institut fur Kraftfahrzeuge
[2]dernehl@embedded.rwth-aachen.de, Lehrstuhl Informatik 11 - Software für eingebettete Systeme
[3]kusmenko@se-rwth.de, Lehrstuhl fur Informatik 3 - Software Engineering

[DDE+17] J. Dankert, C. Dernehl, L. Eckstein, S. Kowalewski, E. Kusmenko, B. Rumpe:
RapidCoop - Robuste Architektur durch geeignete Paradigmen für Kooperativ Interagierende Automobile.
In: Automatisiertes und Vernetztes Fahren (AAET'17), Feb. 2017.
www.se-rwth.de/publications

einrichtungen ermöglichen die Erstellung eines detaillierten Umgebungsmodells und die Kommunikation mit anderen Verkehrsteilnehmern. Es ist absehbar, dass ein Großteil der Verkehrsteilnehmer mit entsprechender Technologie ausgestattet sein wird. Damit sich die Insassen wohl fühlen, muss sich das Fahrzeug jedoch nicht nur sicher im Verkehr bewegen können, seine Handlungen müssen für den Menschen nachvollziehbar sein und mit seinem eigenen Verhalten übereinstimmen. Um dies zu erreichen muss das hochautomatisierte zum kooperativen Fahrzeug erweitert werden. Die optimale und sichere Bewältigung der Herausforderungen kooperativen Fahrens stellt dabei hohe Anforderungen an die Echtzeitfähigkeit und funktionale Sicherheit. Um die Einführung kooperierender Fahrzeuge ressourceneffizient zu ermöglichen, müssen die vorhandene Mittel zudem optimal genutzt werden.

Im Rahmen des Schwerpunktprogramms 1835 der deutschen Forschungsgemeinschaft beschäftigen sich mehrere Institute der RWTH mit verschiedenen Aspekten des kooperativen autonomen Fahrens. Durch Kooperation und Austausch von Informationen kann die Verkehrssicherheit, z.B. durch Kommunikation von Gefahren erhöht werden. Des Weiteren kann das Verkehrsnetz durch kooperatives Planen von Routen effizienter gestaltet werden. Dabei kann die Fairness beim Verteilen von Ressourcen erhöht und die Priorisierung wichtiger Verkehrsteilnehmer wie Feuerwehr- und Krankenwagen sichergestellt werden. Mit dem Konzept der Local Traffic Systems (LTS) stellen wir ein Framework und einen Architekturansatz zur effizienten und strukturierten Verwaltung von kooperativen Verkehrsteilnehmer vor. Dabei untersuchen wir Eigenschaften dieser Verbünde und zeigen auf, wie Komplexität und Aufwand beim Entwurf LTS-basierter Systeme durch den Einsatz von modellbasierter Softwareentwicklung drastisch reduziert werden kann. Durch den Einsatz von Verifikationsmethoden sind wir ferner in der Lage, die Integrität des Gesamtsystems zu jedem Zeitpunkt sicherzustellen.

Des Weiteren können LTS als Abstraktion eines Teilsystems nach außen betrachtet werden. So kann ein LTS, welches von Fahrzeugen einer Kolonne gebildet wurde, durch eine kompakte LTS-Schnittstelle nach außen repräsentiert werden. Muss die Kolonne durch die Entscheidung einer Road Side Unit (RSU) zum Anhalten gebracht werden, muss diese RSU nicht jedes einzelne Fahrzeug der Kolonne ansprechen, sondern lediglich die Schnittstelle des LTS bedienen.

1.1 Stand der Technik

Gruppen autonomer Fahr- oder Flugzeuge – auch Agenten genannt – sind gerade im Bereich der Robotik Bestandteil intensiver Forschung. Hier liegt der Fokus insbesondere auf der Bewegungsplanung für einzelne Agenten oder statische Gruppen von Agenten [ES14]. Die Verwaltung solcher Gruppen, gerade unter Beachtung der Unsicherheit in Bezug auf Sensordaten, das Verhalten anderer Mitglieder und die aktuelle Gruppengröße, stellt weiterhin eine große Herausforderung dar [SGE16]. Viele der Erkenntnisse aus dem Bereich der Robotik werden auch für autonome Fahrzeuge genutzt. [Fre12] stellt mit dem Konzepte der LTS verwandte Ansätze zur Bildung lokaler Verbünde von Verkehrsteilnehmern vor.

Die Entwicklung von Automotive-Software ist dabei von vielen Herausforderungen geprägt, angefangen bei der Heterogenität der verwendeten Plattformen und Protokolle bis hin zur Notwendigkeit der Verifizierbarkeit des entwickelten Codes. Um der Komplexität Herr zu werden, wurden Frameworks zur Standardisierung der Softwareentwicklung für die Automotive-Domäne wie AUTOSAR entwickelt [KBFL+11]. Ferner lässt sich die Komplexität solcher Systeme durch den Einsatz von Architekturbeschreibungssprachen (ADLs) wie AADL und MontiArc beherrschen, welche in erster Linie für die Modellierung statischer Software-Architekturen konzipiert sind [RRW13][FGH06]. Es wurde gezeigt, dass dynamische Architekturen beispielsweise mit Hilfe von Petri-Netzen oder durch den Einsatz von Modes und Mode-Transi-tionen modelliert werden können [CDMR05][FGH06]. Eine weitere Mö-glichkeit der Modellierung bieten zelluläre Automaten, welche insbesondere bei physikalischen Prozessen eingesetzt werden. In der Vergangenheit wurden zelluläre Automaten bereits eingesetzt, um Verkehrssysteme zu modellieren [ES97].

Neben der Modellbildung, spielt die funktionale Sicherheit eine wichtige Rolle bei der Konzeption von Automotive Software. Heutzutage werden inzwischen neben dem Testen auch häufiger formale Methoden zur automatisierten Verifikation von Systemen eingesetzt. Viele der Verifikationtechniken können nicht nur auf der Code Ebene, sondern auch auf Modellen angewendet werden. Jedoch ist hierfür eine formal Beschreibung des Modells notwendig. Contract-based Design [Mey92, EFN+14] ist ein Verfahren, womit Aussagen von Teilsystemen genutzt werden können um Aussagen über das Gesamtsystem treffen zu können. In der Praxis kann Constract-based Design Fehler verhindern, wie z.B. bei der Arine [JM97] da es eine genaue Fehleridentifikation der Komponenten erlaubt, welche Ihre Bedingungen nicht eingehalten haben. Contract-based Design

hat vielfältige Anwendungsmöglichkeiten [SVDP12], u.a. auch Automotive oder Luft und Raumfahrt.

Das Verhalten von einzelnen Fahrzeugen mit Bezug auf deren Trajektorien und Manöver wurde bereits mittels Erreichbarkeitsanalysen erfolgreich online verifiziert[ASB07, AD14]. Im Gegensatz dazu wird in dieser Arbeit die Möglichkeiten zur Verifikation auf Fahrzeuggruppen aufgezeigt.

2 Verteilte Kooperation

Im folgenden Abschnitt werden mögliche Konzepte verteilter Kooperation für Gruppen von Fahrzeugen sowie ein Ansatz für Hierarchisierung vorgestellt. Anschließend wird das Konzept der Local Traffic Systems ausgeführt und Kriterien für eine Bildung formuliert, die durch einen Algorithmus ausgewertet werden. Zuletzt wird die aktuelle Simulationsumgebung und die zugrundeliegenden Modelle beschrieben.

2.1 Konzepte

Grundsätzlich kann zwischen zwei verschiedenen Ausprägungen verteilter Kooperation unterschieden werden: Die Kooperation ist entweder zentral oder dezentral organisiert. Zentrale, d. h. von einem Teilnehmer organisierte, Kooperation bietet den Vorteil, dass die Handlungsplanung und das damit verbundene Treffen von verkehrsrelevanten Entscheidungen im Gegensatz zur dezentralen Ansätzen einfacher und ohne Verhandlung mit den anderen betroffenen Teilnehmern durchgeführt werden kann. Lediglich die Festlegung auf einen organisierenden, bzw. planenden Teilnehmer innerhalb einer Gruppe erfordert eine initiale Verhandlung und – falls notwendig – eine Neuverhandlung sobald sich die Zusammensetzung der Gruppe ändert. Dezentrale Kooperation hingegen benötigt grundsätzlich zusätzliche Verhandlungsphasen, um die individuellen Vorhaben abzustimmen. Gerade spontan veränderliche Situationen im Straßenverkehr limitieren jedoch die Dauer solcher Verhandlungsphasen relativ stark, typischerweise in Bereiche deutlicher unter einer Sekunde. Im Anbetracht der Komplexität heutiger Planungsalgorithmen sind dezentrale Ansätze derzeit nicht, bzw. nur in stark reduzierter Form, echtzeitfähig umsetzbar. Im Folgenden wird deswegen ein zentraler Ansatz vorgestellt.

Zusätzlich wird ein Mehrebenen-Konzept verfolgt, um die Komplexität der notwendigen Planungsvorhaben zu reduzieren. Es wird zwischen drei Ebenen unterschieden:

1. **Verkehrsebene** Auf dieser makroskopischen Ebene findet der Zusammenschluss und die Organisation verteilter Gruppen statt.

2. **Manöverebene** Hier findet die gruppenindividuelle Planung für die jeweiligen Fahrvorhaben statt. Das Ergebnis sind Referenztrajektorien für alle Teilnehmer. Ist das Fahrzeug Teil keiner Gruppe, so plant es Manöver autonom.

3. **Bahnführungsebene** Auf dieser eher mikroskopischen Ebene wird die tatsächliche Bahnführung, d. h. die Verfolgung der Referenztrajektorie individuell von jedem Teilnehmer selbst übernommen.

Der nächste Abschnitt befasst sich mit der Verkehrsebene, insbesondere dem Konzept der LTS.

2.2 Local Traffic Systems

Local Traffic Systems (LTS) sind Ad-Hoc-Verbünde von mobilen Verkehrsteilnehmern und stationären Road Side Units (RSUs), z. B. von intelligenten Ampelsystemen. Sie stellen eine Kommunikationsinfrastruktur für kooperatives Fahrverhalten und -planung bereit. Ein Teilnehmer des LTS übernimmt die Rolle des Koordinators und behält diese bis zu seinem Ausscheiden aus dem LTS oder der Auflösung desselben. Somit ist es sinnvoll, einen Teilnehmer mit möglichst hoher Verweilzeit im LTS und gleichzeitig hoher Rechenleistung zu wählen. Grundlage für die Bildung der LTS an sich sind Kriterien, die es zu definieren gilt. Es soll eine Zusammenschlussverhandlung zweier potentieller Teilnehmer ausgelöst werden, wenn sie kurz- bis mittelfristig eine gegenseitige verkehrstechnische Relevanz besitzen. Beispiele hierfür sind Gefahrensituationen wie eine drohende Kollision, aber auch bevorstehende Annäherung, z. B. aufgrund von Geschwindigkeitsunterschieden oder bei der Anfahrt an eine Kreuzung. Um die verkehrstechnische Relevanz sinnvoll bestimmen zu können, muss auch das reale Straßennetz und die Position der Teilnehmer auf selbigem berücksichtigt werden. So nähern sich z. B. auf der Gegenspur einer Autobahn oder anderweitig baulich getrennten Fahrspuren andere, entgegenkommende Verkehrsteilnehmer zwar regelmäßig sehr stark an, sind aber keineswegs verkehrstechnisch relevant. Abbildung 1 zeigt eine sinnvolle Einteilung in LTS für eine gegebene Verkehrssituation.

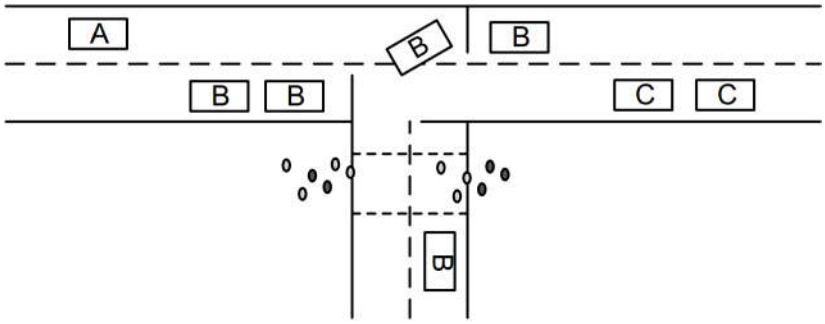

Abbildung 1: Beispielhafte LTS-Einteilung an einer Kreuzung.

Um eine Bewertungsfunktion zu ermöglichen, müssen also Abstandsmaße definiert werden:

- Grundsätzlich sollen Teilnehmer, die zum Zeitpunkt der Bewertung über eine gewisse räumliche Nähe und eine ähnliche Bewegungsrichtung verfügen, einen geringen Abstand d_{dist} besitzen.
- Teilnehmer, deren kommunizierte oder geschätzte Bahn auf dem Straßennetz Schnittpunkte besitzt oder sogar überlappt, sollen einen geringen Abstand d_{traj} besitzen.
- Es soll versucht werden, eine minimale und maximale Gruppengröße über das Abstandsmaß d_{size} einzuhalten.
- Zur Stabilisierung der Gruppeneinteilung sollen LTS zudem mindestens eine gewisse Zeit bestehen bleiben. Es resultiert das Abstandsmaß d_{age}.

Beispielhaft wird im Folgenden das Abstandsmaß d_{size} einer Gruppe G definiert, wobei $[c_{\text{min}}, c_{\text{max}}]$ die angestrebte Gruppengröße angibt:

$$d_{\text{size}}(G) = \begin{cases} (|G| - c_{\text{min}})^2 & \text{für } |G| < c_{\text{min}} \\ 0 & \text{für } c_{\text{min}} \leq |G| \leq c_{\text{max}} \\ (|G| - c_{\text{max}})^2 & \text{für } G| > c_{\text{max}} \end{cases}$$

Vom Ideal abweichende Gruppengrößen werden somit bestraft. Die gewichtete Summe der einzelnen Abstandsmaße ergibt das Gesamtabstandsmaß d_{tot} für ein potentielles LTS. Dieses wird genutzt, um aus

der Menge aller möglichen Einteilungen über alle Verkehrsteilnehmer eine optimale Einteilung im Sinne des Abstandsmaßes zu erhalten. Eine Einteilung wird genau dann als optimal betrachtet, wenn die Summe der Abstände $d_{\text{tot},i}$ für alle enthaltenen Gruppenvorschläge minimal ist. Die optimale Einteilung wird in zyklischen Abständen neu berechnet. Aus den Unterschiedenen zur derzeitigen, tatsächlichen Einteilung werden anschließend die notwendigen Aktionen für den Übergang in die neue Einteilung abgeleitet. Diese Aktionen umfassen das erstmalige Beitreten oder Verlassen von LTS, aber auch den Übergang in andere LTS. Da Anzahl der möglichen Gruppen bei N Fahrzeugen entspricht der Potenzmenge 2^N abzüglich der leeren Menge. Da eine Gruppeneinteilung zusätzlich noch aus Kombinationen von Gruppen besteht, ist die Menge der möglichen Einteilungen noch größer. Eine Berechnung aller Einteilungen ist somit mit einem Rechenaufwand verbunden, der nicht mehr in Echtzeit darstellbar ist. Algorithmus 1 repräsentiert einen gierigen Algorithmus, der schnell eine gute Gruppeneinteilung findet, ohne jedoch die Optimalität garantieren zu können.

Algorithmus 1 Gieriger Algorithmus zur Gruppeneinteilung.

```
1:  function CALC_LTS(V = {v_1, ..., v_n})
2:      E_0 ← {{v_1}, ..., {v_n}}          ▷ Starte mit Einzahlfahrzeug-Gruppen
3:      SORTEDMAP_ADD(d(E_0), E_0)                  ▷ Speichere Bewertung
4:      while not terminated do
5:          E ← SORTEDMAP_GET_BEST()
6:          for all G_i ∈ E do                      ▷ Teste alle Kombinationen
7:              for all G_j ∈ E, G_i ≠ G_j do
8:                  E' ← E \ {G_i, Gj} ∪ {G_i ∪ G_j}
9:                  if ¬SORTEDMAP_CONTAINS(E') then  ▷ Unbekannt?
10:                     d(E') ← d(E) − d(G_i) − d(G_j) + d(G_i ∪ G_j)
11:                     SORTEDMAP_ADD(d(E'), E')
12:                     if d'(E') < d(E) then
13:                         E ← E'                  ▷ Neue beste Einteilung
14:                     end if
15:                 end if
16:             end for
17:         end for
18:     end while
19: end function
```

Der Algorithmus startet mit einer Einteilung, in der jede Gruppe aus einem Teilnehmer besteht. Mit jedem Schritt werden alle Möglichkeiten für Kombinationen aller bis dahin bestehenden Gruppen der bisher besten Einteilung evaluiert. Um das Verfahren weiter zu beschleunigen werden Gruppen, deren Teilnehmer ohnehin ein zu hohes Abstandsmaß d_{dist} aufweisen, direkt verworfen, ohne die anderen Maße zu evaluieren. Zusätzlich wird Caching eingesetzt, um einmal berechnete Gruppengesamtabstandsmaß $d_{\text{tot},i}$ nicht unnötig erneut berechnen zu müssen. Die berechneten Einteilungen werden in einer Liste sortiert, an der Spitze steht stets die bisherige optimale Einteilung. So ist zudem möglich, einen individuellen Tradeoff zwischen Rechenzeit und Optimalität des Ergebnis zu erzielen. Es ist meist mit weniger als zehn Iterationen möglich, eine sinnvolle Verteilung zu finden, wie auch die Ergebnisse der Simulation im nächsten Abschnitt zeigen.

2.3 Hierarchisierung

In Situationen mit hohen Verkehrsdichten kann eine weitere Ein- bzw. Unterteilung die Komplexität reduzieren. Mit hierarchischen LTS können mehrere LTS in einem weiteren, übergeordneten LTS zusammengefasst werden. Die Kommunikation mit den Teilnehmern der untergeordneten LTS erfolgt hierbei über die jeweiligen LTS-Koordinatoren. Dieser Ansatz ist hilfreich, um die Komplexität von Planungs- und Verifikationsvorhaben einzugrenzen und damit die Echtzeitigfähigkeit zu erhalten.

2.4 Simulationsumgebung

Um die entworfenen Algorithmen zu evaluieren, wurde eine einfache, zeitdiskrete Simulationsumgebung zur Berechnung und Visualisierung eingesetzt, angelehnt an zelluläre Automaten. Grundlage der Simulation ist zunächst das Kartenmaterial, das in Form von gerichteten Graphen abgespeichert und verwaltet wird, vergleichbar zu [BZS14]. Dieses kann künstlich erzeugt oder aus automatisiert aus realen Kartendaten, wie sie OpenstreetMap zur Verfügung stellt, generiert werden und unterstützt somit somit eine Vielzahl an Verkehrsszenarien. Vergleichbar zu dem oben beschriebenen Mehrebenen-Konzept werden wie in Abbildung 2 dargestellt auch die Kartendaten unterteilt.

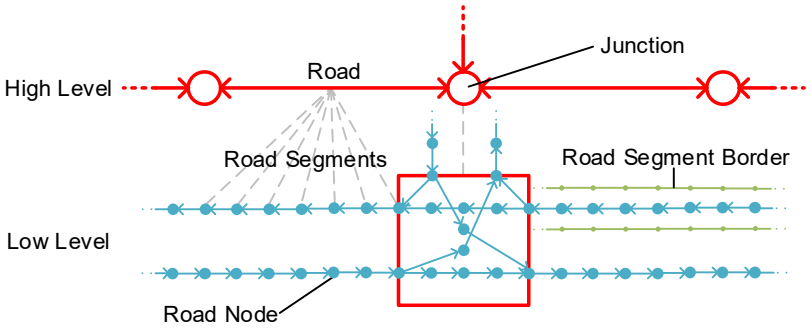

Abbildung 2: Kartendaten mit unterschiedlichen Detailebenen

Die oberste Ebene gibt das tatsächliche Straßennetz nur grob wieder. Informationen wie Kreuzungen, Abzweigungen oder die Richtgeschwindigkeit sind enthalten, Detailinformationen wie Spuren fehlen jedoch. Dieses Kartenmaterial ist somit ideal für Abstandsschätzungen oder Routing. Eine weitere Ebene enthält die tatsächlichen, detaillierter abgelegten Spuren, mit denen eine Manöver- oder Bahnführungsplanung durchgeführt werden kann. Die Daten sind vergleichbar mit den Eingangs vorgestellten Lanelets und grundsätzlich kompatibel. Die zusammengehörigen Elemente der beiden Ebenen sind zusätzlich jeweils miteinander verknüpft. Da bisher untersuchten Ansätze nicht auf der Bahnführungsebene stattfinden, können die Eigenschaften und das Verhalten der mobilen Teilnehmer gut über einfache Modelle angenähert werden. So werden die Fahrzeuge als Punktmassen angesehen, die sich gemäß ihrer aktuellen Geschwindigkeit entlang der Kanten des Straßengraphen bewegen. Wechselt die Richtgeschwindigkeit oder soll z.B. abgebogen werden, wird die Geschwindigkeit durch Bremsen oder Beschleunigen entsprechend angepasst. Diese Eigenschaften sind vergleichbar mit Konzept der Petri-Netze. Die Fahrzeuge innerhalb der Simulation verfolgen entweder ein spezielles Ziel oder entscheiden bei jeder Abbiegemöglichkeit zufällig. Dieses Verhalten wird für einen gewissen Zeithorizont vorausberechnet und die resultierende fahrzeugindividuelle, geplante Trajektorie nach außen kommuniziert. Diese steht somit ebenso wie die aktuelle Position und Geschwindigkeit jedem anderen Teilnehmer in Reichweite zur Verfügung. Den Teilnehmern stellen also alle Daten zur Verfügung, die für eine LTS-Einteilung benötigt werden. Abbildung 3 zeigt exemplarisch die resultierenden Ein-

teilung für verschiedene Fahrvorhaben der Fahrzeuge in derselben simulierten Situation, einem Kreisverkehr innerhalb Aachens.

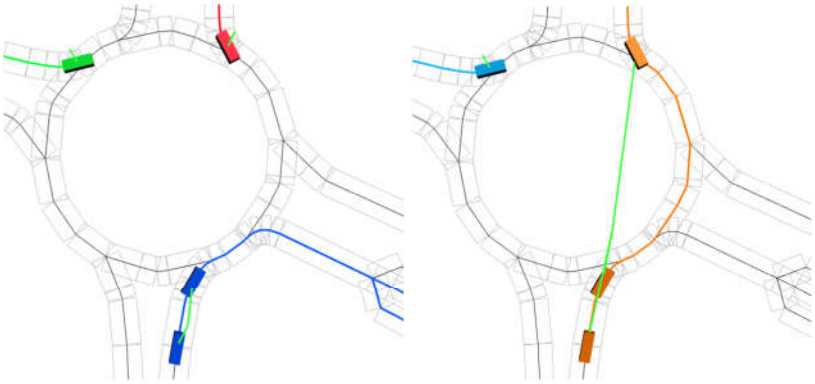

Abbildung 3: LTS-Einteilung für verschiedene Fahrvorhaben. Farbe und die Verbindungslinie zeigen eine gemeinsame LTS-Zugehörigkeit an.

3 Modellbasierte LTS

Zur Beschreibung der vorgestellten LTS-Systeme wird ein modellbasierter Ansatz verwendet. Ziel modellbasierter Softwareentwicklung (MBSE) ist es, domänenspezifische Logik von konkreten Implementierungsdetails zu entkoppeln. Zum einen werden dadurch Systemstruktur und -verhalten durch eine geeignete Modellierungsform explizit beschrieben und somit für Domänenexperten zugänglich gemacht. Zum anderen reduziert dieser Ansatz Entwicklungskosten, da ein Großteil des Codes mit Hilfe von Generatoren automatisiert aus den Modellen erzeugt werden kann [KRV10]. Insbesondere in hochgradig heterogenen Systemen wie dem kooperativen Fahren, bei denen Programmcode für eine Vielzahl unterschiedlicher Systeme angepasst werden muss, kann durch den Einsatz von MBSE ein Großteil des Portierungsaufwandes durch Code-Generierung automatisiert werden. Hierfür muss für jede Zielplattform einmalig ein Code-Generator bereitgestellt werden. Ferner erlauben formale Modelle den Einsatz formaler Verifikationsmethoden, welche nicht ohne Weiteres auf in einer General Purpose Language (GPL) geschriebenem Programmcode verwendbar sind. Auch lassen sich modellbasierte Methoden mit agilen Prozessen kombinieren[Rum12].

Zur Modellierung der LTS-Systemarchitektur benutzen wir die auf dem Komponenten- und Konnektor-Paradigma (C&C) basierende Spra-

che MontiArc [HRR12]. Die zentralen Elemente des Metametamodells dieses Paradigmas sind, wie der Name es sagt, Komponenten und Konnektoren. Unter einer Komponente ist dabei ein abgeschlossenes System zu verstehen, welches für die Kapselung einer seiteneffektfreien Aufgabe zuständig ist. Zur Ausführung kann eine Komponente Eingabeparameter über getypte Eingangsports entgegennehmen und die Ergebnisse der Berechnung über ebenfalls getypte Ausgangsports ausgeben. Durch verbinden von Ausgangsports mit Eingangsports anderer Komponenten können somit streng getypte Datenflüsse definiert werden. Die von einer Komponente übernommene Aufgabe kann je nach zu modellierender Abstraktionsebene die komplette Armsteuerung eines Roboters oder eine simple logische Verknüpfung sein. Über beliebig tiefe hierarchische Verschachtelungen können verschiedene Abstraktionsebenen miteinander verknüpft werden. Wie oft und wann eine Komponente auszuführen ist, muss durch das zugrunde liegende Zeitmodell bzw. eine implizit oder explizit gegebene Ausführungsreihenfolge definiert werden.

In Industrie und Forschung etablierte C&C-basierte Sprachen sind zumeist proprietäre grafische Lösungen wie Simulink und LabView, welche mit Fokus auf eine bestimmte Domäne wie Regelungs- oder Messtechnik zugeschnitten sind. Im Gegensatz dazu ist MontiArc dank der MontiCore-Technologie eine erweiterbare Modellierungssprache mit textueller Syntax[KRV10]. So kann MontiArc durch geeignete Erweiterungen an neue Domänen, wie in diesem Fall das kooperative Fahren, angepasst werden. Die Modelle sollen anschließen verwendet werden, um eine ROS-kompatible Code-Basis zu generieren. Ansätze zur Entwicklung Cyberphysikalischer Systeme mit Hilfe von MontiArc wurden beispielsweise in [RRW13][RRRW14] diskutiert. Auch existieren effiziente Algorithmen zur Verifikation struktureller Eigenschaften von C&C-Modellen[MRR14], was für den Einsatz in sicherheitskritischen Systemen von großer Bedeutung ist. In Abbildung 4 ist ein C&C-Modell für ein Szenario mit drei Verkehrsteilnehmern, welche insgesamt drei LTS angehören, dargestellt. Dabei kapseln die Participant-Komponenten weitestgehend die Logik eines autonomen Fahrzeugs, während die LTS-Komponenten die Koordinationsaufgaben übernehmen. Es sei zu beachten, dass dieses Modell keinerlei Annahmen über die Verteilung der Komponenten trifft. Die LTS-Komponenten können durchaus auf der selben Hardware laufen, wie eine von diesen gesteuerte Participant-Komponente. Alle Verbindungen sind als logische Ende-zu-Ende-Verbindungen zu verstehen und alle Datenflüsse beziehen sich auf domänenrelevante Informationen. Die Umsetzung logischer Verbindungen obliegt der darunter liegenden Netzwerk-

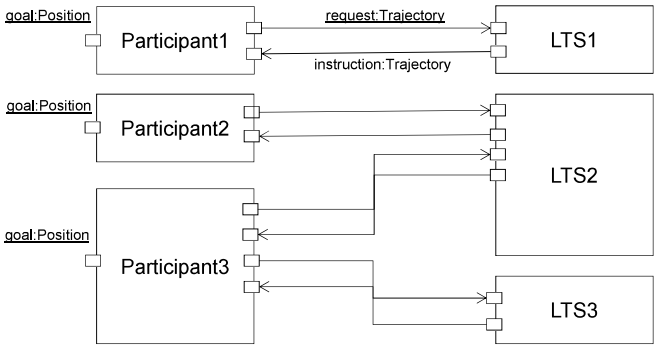

Abbildung 4: C&C-Modell eines hierarchischen LTS-Systems

technologie. Somit handelt es sich bei LTS um ein Konzept, welches den OSI-Schichten 5-7 zuzuordnen ist, und es wird von der technischen Realisierung der notwendigen Kommunikationsinfrastruktur abstrahiert. Ferner können die dargestellten Komponenten eine beliebig tiefe, endliche Subkomponentenhierarchie enthalten, was hier aus Platzgründen nicht dargestellt wird.

In diesem Beispiel befindet sich Teilnehmer 1 in seinem eigenen LTS ohne weitere Teilnehmer. Teilnehmer 2 und 3 sind Mitglieder eines gemeinsamen LTS und können somit Informationen untereinander austauschen. Die beiden LTS sind ferner über ein Super-LTS miteinander verbunden und können über die LTS-Schnittstellen miteinander kommunizieren. Teilnehmer 3 ist außerdem Mitglied in einem dritten LTS. Um unnötige Komplexität zu vermeiden, sind in der Abbildung nur zwei Arten von Datenflüssen, jeweils vom Typ *Trajectory*, dargestellt: Verkehrsteilnehmer können zum einen Trajektorien-Anfragen an die zuständigen LTS senden, zum anderen können sie gegebenenfalls modifizierte Trajektorien-Anweisungen von diesen erhalten. Selbstverständlich muss ein vollständiges Modell weitere Ports und Konnektoren beinhalten, um ein voll funktionsfähiges System realisieren zu können.

Die textuelle Definition des in Abbildung 4 betrachteten Modells ist in Listing 1 gegeben. Hier werden in den Zeilen 1-8 zunächst die Komponententypen *Participant* und *LTS* definiert. Beide Komponententypen verfügen ferner über je einen Konfigurationsparameter des Typs *int*. Dieser gibt an, wie viele Kommunikationspartner die Komponente haben soll und somit wie lang die Ein- und Ausgangsport-Arrays sein müssen.

```
1  component Participant(int n)  {                                    MA
2    port
3      in Position goal, in Trajectory instruction[n], out
           Trajectory request[n];
4  }
5  component LTS(int m)  {
6    port
7      in Position goal , in Trajectory request[m] , out
           Trajectory instruction[m];
8  }
9
10 component Participant participants[3]([1;1;2]);
11 component LTS lts[3]([1;2;1]);
12
13 connect participants[:].request[1] -> lts[:].request[1];
14 connect lts[:].instruction[1] ->
15     participants[:].instruction[1];
16
17 connect participants[3].request[2] -> lts[2].request[2];
18 connect lts[2].instruction[2] ->
19     participants[3].instruction[2];
```

Listing 1: Textuelle Modellierung von LTS in MontiArc.

Unsere MontiArc-Erweiterung unterstützt die Erstellung von Komponentenarrays und bietet eine kompakte Syntax zu deren Verknüpfung an. So werden alle Komponenten des Modells in den Zeilen 10 und 11 instantiiert. Dabei werden die Konfigurationsparameter bei einer Array-Initialisierung als Liste angegeben, welche für jede Komponente einen individuellen Parameter bereithält.

Die statische Verknüpfung der Komponenten erfolgt schließlich in den Zeilen 13-19. Zum einen können dabei Ports einzeln miteinander verbunden werden, wie dies zum Beispiel in den Zeilen 17 und 18 geschieht. Zum anderen bietet unsere MontiArc-Erweiterung Kurzschreibweisen wie den Doppelpunktoperator, welcher alle Modellelemente des Arrays selektiert. Der Konnektor verbindet dann jedes ausgewählte Element der linken Seite mit demjenigen Element auf der rechten Seite, welches den selben Array-Index besitzt. Für komplexere Verschaltungen können ferner logische Ausdrücke verwendet werden.

Ein solches Modell definiert noch kein Verhalten für seine Komponenten. Dieses kann auf verschiedene Weisen spezifiziert werden:

Verwenden einer Komponentenbibliothek: So wie es für viele Sprachen üblich ist, können auch in MontiArc vorgefertigte Bibliotheken eingebunden und verwendet werden.

Einbettung domänenspezifischer Sprachen: MontiCore verfügt über Kompositionsmechanismen, welche es erlauben, Verhaltenssprachen in das Modell einzubetten [KRV10]. Im Rahmen dieses Projekts sollen Zustandsautomaten sowie mathematische Ausdrücke für Verhaltensdefinitionen eingesetzt werden.

Handgeschriebener Code: In einigen Fällen kann es sinnvoll sein, aus der Architektur-Beschreibung lediglich ein Code-Gerüst generieren zu lassen und die notwendige Verhaltensbeschreibung von Hand zu implementieren. Bei diesem Ansatz gehen Vorteile von MBSE verloren, da Code für jede Zielplattform entwickelt werden muss.

Das hier vorgestellte Modell kann sich basierend auf den Definitionen aus Abschnitt 2 im Laufe der Zeit ändern, da Fahrzeuge LTS verlassen und diesen beitreten können. Auch können LTS ganz aufgelöst oder neu erstellt werden. Konzepte zur dynamischen Erstellung von Komponenten und Konnektoren und deren Rekonfiguration zur Laufzeit sind Bestandteil weiterer Untersuchungen.

4 Verifikation

Die Realisierung von LTS in Beispielszenarien erfordert eine Validierung und Verifikation des Systems. Im Folgenden werden zunächst die Sicherheitsziele erläutert und der Zustandsraum des Systems beschrieben. Danach werden Konzepte zur Einschränkung des Zustandsraums präsentiert, welche eine effiziente Verifikation ermöglichen. Mit einer automatisierten Verifikation soll sichergestellt werden, dass vorgegebene Sicherheitsziele durch das System erfüllt werden. In Rahmen von LTS ist damit Kollisionsfreiheit von Teilnehmern, anderen LTS und Infrastruktur gemeint. Der Zustand des Systems ist die Position und Geschwindigkeit jedes Teilnehmers, sowie die Zusammensetzung der LTS. Da das System potentiell ewig läuft, ist auch der Zustandsraum unendlich groß. Auf Kosten der Genauigkeit können, mithilfe von Abstraktionen, die Zustände reduziert werden. So kann z.B. die Geschwindigkeit von Fahrzeugen als Intervall abgebildet werden, wodurch direkt mehrere Szenarien für verschiedene Geschwindigkeiten beschrieben werden. Doch selbst durch diese einfachen Abstraktionen ist eine automatisierte Analyse des Systems grundsätzlich

rechenaufwändig. Deshalb wird der Aufwand für die automatisierte Verifikation durch weitere Konzepte reduziert.

4.1 Abstraktionstechniken

Durch die Beschränkung des zeitlichen Horizont kann der Zustandsraum eingeschränkt werden, sodass eine automatisierte Verifikation durchgeführt werden kann. Ähnliche Ansätze existieren bereits z.B. bounded model checking. Jedoch können so nur Aussagen über den gewählten Zeitraum gemacht werden.

In Abbildung 5 sind insgesamt drei Zeitschritte des Systems abgebildet. Innerhalb eines diskreten Zeitschrittes ändern sich die Positionen, sowie mögliche Routen der einzelnen Teilnehmer und LTS. Mit der Zustandsübergangsfunktion kann der gesamte Zustandsraum für diese Zeitschritte erfasst und analysiert werden.

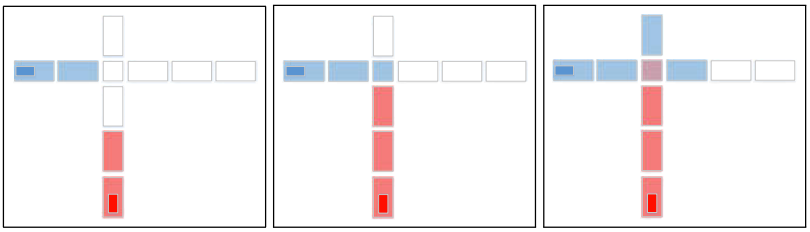

Abbildung 5: Verifikation für drei Zeitschritte.

Weiterhin kann der Zustandsraum durch eine lokale Betrachtung weiter eingeschränkt werden, sodass Aussagen über den betrachteten Raum gemacht werden können. Insbesondere bei großen Netzen ist eine weitere örtliche Einteilung sinnvoll um den Berechnungsaufwand gering zu halten.

Eine weitere Möglichkeit den Zustandsraum zu reduzieren ist eine Überapproximation der Taktung, d.h. die Zusammenlegung mehrere Zeitschritte zu zwei Abstrakten Zuständen. Wenn sich z.B. die Geschwindigkeit für ein LTS innerhalb eines Intervalls bewegt, kann der Verlauf des LTS für mehrere Zeitschritte berechnet werden und direkt erreichbare Zustände identifiziert werden. Beispielsweise sei ein LTS mit Geschwindigkeit von 1-3 Zellen pro Zeitschritt unterwegs, so kann man bereits vorberechnen, dass sich das LTS nach n Schritten sich um $[n, 3n]$ Zellen weiterbewegt hat.

4.2 Contract-based Design

Jede Komponente wird durch Vorbedingungen und Nachbedingungen oder auch Annahmen und Garantien beschrieben, wobei die Nachbedingungen gelten, wenn die Vorbedingungen erfüllt sind. Beispielsweise kann ein LTS garantieren an einer bestimmten Position zu sein, wenn keine Kollision mit einem anderen LTS stattfindet. Da Vorbedingungen und Nachbedingungen boolesche Aussagen sind, d.h. wahr oder falsch sind, wird so die Systemanalyse zunächst vereinfacht. Sobald eine Komponente eine Vorbedingung erfüllt, gelten alle Nachbedingungen, sodass die Vorbedingungen als Disjunktion und die Nachbedingungen als Konjunktion dargestellt werden können. Eine solche Systembeschreibung erlaubt es verbundene Systeme mit den Vorbedingungen und Nachbedingungen zu analysieren. Jedoch müssen in dieser Darstellung die Bedingungen aller Komponenten unabhängig von einander sein, sodass im Bezug auf LTS Design by Contract (DbC) für einzelne LTS nur zeitlich begrenzt eingesetzt werden kann. Insbesondere wenn vorher garantiert werden kann, dass zwei LTS ausreichend von einander für eine bestimmte Anzahl von Zeitschritten entfernt sind. Lässt sich beispielsweise automatisiert belegen, dass sich die Pfade zweier LTS bis zu einem festen Zeitpunkt nicht schneiden, so kann daraus direkt Kollisionsfreiheit von zwei Teilnehmern eines jedes einzelnen LTS abgeleitet werden. Jedoch kann mit der genannte Aussage zunächst nichts über Kollisionen innerhalb der einzelnen LTS ausgesagt werden.

Neben einer weiteren Perspektive auf das Gesamtsystem, bietet DbC auch die Möglichkeit den Rechenaufwand zu reduzieren So können Ergebnisse der Verifikation für einzelne LTS mit DbC wiederverwendet werden, da die Vorbedingungen und Nachbedingungen weiterhin gültig sind.

4.3 Parallelisierung und Online Verifikation

Ein weiterer Aspekt um den zeitlichen Aufwand der Verifikation zu reduzieren ist die Parallelisierung, d.h. die parallele Verifikation der Teilsysteme. Ähnlich wie bei DbC, muss auch hier eine Unabhängigkeit der einzelnen Teilsysteme sichergestellt werden. Durch die räumliche Trennung für einen begrenzten Zeithorizont kann eine Kollisionserkennung für einzelne Bereiche parallel berechnet werden. Am Ende müssen die Ergebnisse der einzelnen Bereiche zusammengesetzt werden um so die Einflüsse zusammenhängende Bereiche betrachten zu können. Dabei kann es vorkommen, dass sich die Bereiche für die parallele Analyse sich während des Verlaufs wachsen, schrumpfen oder sich verschieben. Ein weiterer Vorteil bei der Aufteilung des Problems ist die Dezentralisierung, denn

bisher wurde die Verifikation in einem zentralistischer Kontext diskutiert. Jedoch lassen sich viele der Methoden auch dezentral hierarchisch auf die einzelnen LTS anwenden.

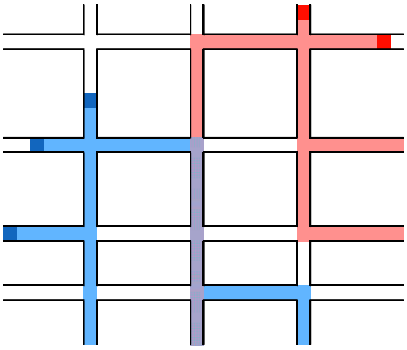

Abbildung 6: Zwei LTS in einem Straßennetz

In Abbildung 6 sind zwei LTS in blau und rot dargestellt. Teilt man das Straßensystem vertikal in der mittleren Straße, so erreicht man eine Partitionierung der LTS im Bezug auf den aktuellen Zustand. Die potentiellen Pfade der LTS überschneiden sich jedoch, sodass sich die LTS auch beeinflussen können. Dennoch kann die Berechnung bis zu Neuordnung der LTS oder Erreichen der mittleren Straße parallel berechnet werden. Danach müssen die Prozesse und Ergebnisse synchronisiert werden und die Verifikation in den neuen Zustand überführt werden.

5 Zusammenfassung

In Abschnitt 2 wurde ein Ansatz zur Organisation von verteilter Kooperation vorgestellt, die sogenannten Local Traffic Systems. Diese wurden definiert und ein echtzeitfähiger Algorithmus zur effizienten Einteilung präsentiert. Die unter anderem für die Validierung dieses Algorithmus genutzte Simulationsumgebung wurde beschrieben und erste Ergebnisse präsentiert. Zukünftige Arbeiten im Kontext der LTS umfassen die ausführliche Behandlung der aus dem Algorithmus resultierenden Veränderung, also Veränderungen der LTS-Mitgliedschaften zur Laufzeit. Hier gilt es insbesondere, den Einfluss auf laufende Planungsvorhaben auf der Manöverebene zu untersuchen.

In Abschnitt 3 wurde gezeigt, dass ein modellbasierter Ansatz zur Entwicklung von LTS-Architekturen viele Vorteile im Vergleich zu

konventioneller Software-Entwicklung bietet. Dafür verwenden wir das Komponenten- und Konnektor-Paradigma und eine Erweiterung der Architekturbeschreibungssprache MontiArc. In Zukunft sind Spracherweiterungen zu untersuchen, welche durch geeignete Konzepte das Modellieren dynamischer Veränderungen des Systems zur Laufzeit ermöglichen. Damit sollen LTS-Operationen, wie sie in Abschnitt 2 diskutiert wurden, explizit in MontiArc beschrieben werden können. Insbesondere ist die Beschaffenheit der Auslöser für das Erstellen und Zerstören von Komponenten und Konnektoren sowie Konzepte für erweiterbare Komponenten-Schnittstellen zu analysieren. Weiterhin wurde in diesem Papier aufgezeigt, dass neben der Modellbildung auch der funktionale Sicherheit eine wichtige Rolle in diesem Projekt zuteil wird. Mittels Design by Contract und können Systeme formalisiert und verifiziert werden.

Literatur

[AD14] Matthias Althoff and John M Dolan. Online verification of automated road vehicles using reachability analysis. *IEEE Transactions on Robotics*, 30(4):903–918, 2014.

[ASB07] Matthias Althoff, Olaf Stursberg, and Martin Buss. Online verification of cognitive car decisions. In *2007 IEEE Intelligent Vehicles Symposium*, pages 728–733. IEEE, 2007.

[BZS14] Philipp Bender, Julius Ziegler, and Christoph Stiller. Lanelets: Efficient map representation for autonomous driving. In *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pages 420–425. IEEE, 2014.

[CDMR05] Lawrence Cabac, Michael Duvigneau, Daniel Moldt, and Heiko Rölke. Modeling dynamic architectures using nets-within-nets. In *International Conference on Application and Theory of Petri Nets*, pages 148–167. Springer, 2005.

[EFN+14] H-Christian Estler, Carlo A Furia, Martin Nordio, Marco Piccioni, and Bertrand Meyer. Contracts in practice. In *International Symposium on Formal Methods*, pages 230–246. Springer International Publishing, 2014.

[ES97] J. Esser and M. Schreckenberg. Microscopic simulation of urban traffic based on cellular automata. *International Journal of Modern Physics C*, 08(05):1025–1036, 1997.

[ES14] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[FGH06] Peter H Feiler, David P Gluch, and John J Hudak. The architecture analysis & design language (aadl): An introduction. Technical report, DTIC Document, 2006.

[Fre12] Christian Frese. *Planung kooperativer Fahrmanöver für kognitive Automobile*, volume 10. KIT Scientific Publishing, 2012.

[HRR12] Arne Haber, Jan Oliver Ringert, and Bernard Rumpe. MontiArc - Architectural Modeling of Interactive Distributed and Cyber-Physical Systems. Technical Report AIB-2012-03, RWTH Aachen, february 2012.

[JM97] J-M Jazequel and Bertrand Meyer. Design by contract: The lessons of ariane. *Computer*, 30(1):129–130, 1997.

[KBFL+11] Frank Kirschke-Biller, S Fürst, S Lupp, S Bunzel, S Schmerler, R Rimkus, et al. Autosar–a worldwide standard current developments, roll-out and outlook. In *15th International VDI Congress Electronic Systems for Vehicles, Baden-Baden, Germany*, 2011.

[KRV10] Holger Krahn, Bernhard Rumpe, and Steven Völkel. Monticore: a framework for compositional development of domain specific languages. In *International Journal on Software Tools for Technology Transfer (STTT)*, 2010.

[Mey92] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10):40–51, 1992.

[MRR14] Shahar Maoz, Jan Oliver Ringert, and Bernhard Rumpe. Verifying Component and Connector Models against Crosscutting Structural Views. In *36th International Conference on Software Engineering (ICSE 2014)*, pages 95–105, Hyderabad, India, June 2014. ACM New York.

[RRRW14] Jan Oliver Ringert, Alexander Roth, Bernhard Rumpe, and Andreas Wortmann. Code Generator Composition for Model-Driven Engineering of Robotics Component & Connector Systems . In *1st International Workshop on Model-Driven Robot Software Engineering (MORSE 2014)*, volume 1319 of *CEUR*

Workshop Proceedings, pages 66 – 77, York, Great Britain, July 2014.

[RRW13] Jan Oliver Ringert, Bernhard Rumpe, and Andreas Wortmann. MontiArcAutomaton: Modeling Architecture and Behavior of Robotic Systems. In *Workshops and Tutorials Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 6-10 2013.

[Rum12] Bernhard Rumpe. *Agile Modellierung mit UML : Codegenerierung, Testfälle, Refactoring*. Springer Berlin, 2nd edition, June 2012.

[SGE16] Tina Setter, Andrea Gasparri, and Magnus Egerstedt. Trust-based interactions in teams of mobile agents. In *American Control Conference (ACC), 2016*, pages 6158–6163. IEEE, 2016.

[SVDP12] Alberto Sangiovanni-Vincentelli, Werner Damm, and Roberto Passerone. Taming dr. frankenstein: Contract-based design for cyber-physical systems. *European journal of control*, 18(3):217–238, 2012.