$\mathcal{H}$esperia: Framework zur Szenario-gestützten Modellierung und Entwicklung Sensor-basierter Systeme

Christian Berger und Bernhard Rumpe
Rheinisch-Westfälische Technische Hochschule Aachen
Lehrstuhl für Software Engineering
Ahornstraße 55, 52074 Aachen, www.se-rwth.de

Abstract: Moderne eingebettete Systeme durchdringen nicht mehr nur industrielle Domänen der Automatisierungstechnik, sondern beispielsweise in Form zunehmend intelligenterer Fahrerassistenzsysteme auch Domänen, in denen sie direkt in Wechselwirkung mit dem Konsumenten und ihrer Umwelt agieren. Die verwendeten Algorithmen werden zwar häufig durch Methoden des *virtual prototyping* bereits im Labor getestet, doch bei komplexen Systemen zur Sensor-basierten Wahrnehmung sind Laborversuche nicht mehr ausreichend und aufwändige Erprobungen im Feld mit unterschiedlichen Szenarien daher erforderlich. Dieser Beitrag beschreibt ein Framework zur Plattform-übergreifenden, Szenario-gestützten Modellierung und Entwicklung Sensor-basierter Systeme mittels Zeit- und Energie-schonender Laborversuche.

Keywords: Automotive Systems, Modellierung, Domänen-spezifische Sprachen, Simulation

1 Einführung und Motivation

Die Qualitätssicherung eingebetteter Systeme und kürzere Entwicklungszeiten bei gleichzeitig steigender Systemkomplexität erfordern Entwicklungswerkzeuge, die über die Unterstützung bei der Systementwicklung hinaus auch den Systemkontext adäquat in den Entwicklungsprozess mit einbeziehen [Bro05, Und07]. Für ausschließlich virtuelle Produkte, wie beispielsweise auf Java basierende, verteilte Anwendungen, die im Webbrowser ausgeführt werden können, gibt es eine Vielzahl ausgereifter Werkzeuge, wie die Ecplise Test and Performance Tools Platform [The09b], mit denen Daten des Systemkontextes für die Entwicklung eines Systems bereitgestellt werden können. Entwicklungswerkzeuge für Umfeld-wahrnehmende Systeme oder *cyber physical systems*, die mittels Sensoren ihre Umwelt erfassen und mit ihr interagieren, um ressourcenschonende aber dennoch detaillierte Erprobungen bereits im Labor zu erlauben, indem sie Systemsimulationen in den Fokus des Entwicklungsprozesses rücken und durch ein einheitliches Werkzeug anbieten, sind bislang nur unzureichend verfügbar.

Dieser Beitrag beschreibt die Szenario-gestützte Modellierung des Systemkontextes und seine Nutzung innerhalb des Frameworks $\mathcal{H}$esperia[1] zur Entwicklung Sensor-basierter

[1] Der Name "$\mathcal{H}$esperia" leitet sich von einem Ort in Kalifornien ab, an dem das Team CarOLO während der DARPA Urban Challenge 2007 [RBL+08] untergebracht war.

Systeme. Besonderer Fokus dieses Frameworks liegt auf der Generierung von Rohdaten für virtuelle Sensoren zur Umweltwahrnehmung basierend auf Modellen des Systemkontextes. Die Arbeit ist dazu wie folgt gegliedert: Abschnitt 2 wird die textuelle sowie graphische Modellierung des Systemkontextes zur Nutzung als *single-point-of-truth* (SPOT) im Rahmen der Systementwicklung beschreiben. Im Abschnitt 3 werden relevante Aspekte der Software-Architektur des Frameworks beschrieben; die Nutzung der Modelle für virtuelle Sensoren wird im Abschnitt 4 erläutert. Dieser Beitrag wird gegenüber verwandeter Arbeiten im Abschnitt 5 abgegrenzt, bevor die Arbeit im Abschnitt 6 endet.

2 Modellierung des Systemkontextes

Regressionstests-basierende Qualitätssicherung diskreter Algorithmen wird heutzutage mit Unit-Test-Frameworks wie etwa CxxTest [Tig09] durchgeführt. Hierbei wird der Systemkontext eines Algorithmus durch eine endliche Menge von konstanten Eingabedaten und erwarteten Ausgabedaten modelliert, die mit den berechneten Ergebnissen des Algorithmus verglichen werden, um die Resultate der Berechnung eines Algorithmus für eine konkrete Eingabe zu validieren. Die Überprüfung wird durch geeignete Metriken, wie beispielsweise Anweisungs- oder Zweigüberdeckung zur Bildung geeigneter Äquivalenzklassen für die Eingabedaten ergänzt [Kan95]. Dieses Verfahren hat sich zum Beispiel zur Qualitätssicherung und zum *virtual prototyping* eines Algorithmus zur Diskretisierung der durch eine Sensordatenfusion bereitgestellten Fahrzeugumwelt auf abstrakter Ebene als geeignet erwiesen [BBKR09].

Rückt jedoch die Qualitätssicherung und das *virtual prototyping* der Umfeldwahrnehmung selbst in den Fokus, steigt die Komplexität der Eingabedaten aufgrund der Menge und Abhängigkeit von der Zeit stark an. Hierbei wird der Entwickler eines Sensordatenverarbeitungssystem durch Einsatz eines Unit-Test-Frameworks nicht mehr ausreichend unterstützt, da die Modellierung eines geeigneten Systemkontextes zu aufwändig ist.

2.1 Definition einer domänenspezifischen Sprache

Alternativ bietet es sich hierfür an, eine ausreichend umfangreiche Menge an Messfahrten mit dem Sensoren-Träger durchzuführen, um ein möglichst breites Spektrum unterschiedlicher Situationen aufzuzeichnen. Hierbei ist allerdings die Wiederholbarkeit einer spezifischen Situation für unterschiedliche Sensorkonfigurationen sowie die vollständige Kontrolle aller Umgebungsparameter nicht immer möglich. Darüber hinaus lässt sich mit diesem Vorgehen nur der Bereich der Sensor-basierten Wahrnehmung unterstützen, da dieses Verfahren keine Rückkopplung erlaubt (*open-loop*-Tests). Daher ist es sinnvoll, das Verfahren zur Generierung der benötigten Eingabedaten nachzubilden (siehe Abschnitt 4) und den zu betrachtenden Systemkontext in der erforderlichen Detaillierung zu modellieren, um die benötigten Daten synthetisch in das System einzubringen und dadurch Rückkopplungen (*closed-loop*-Tests) zu ermöglichen.

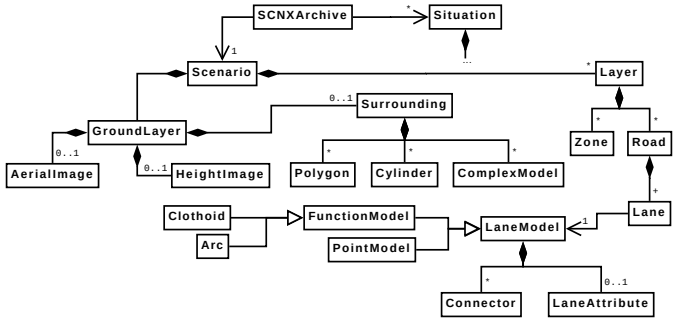


Abbildung 1: Ausschnitt des statischen Systemkontextes für eine automotive Anwendung.

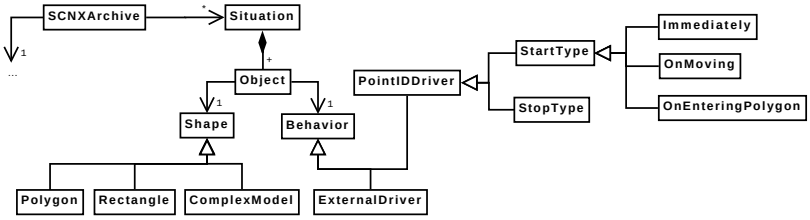
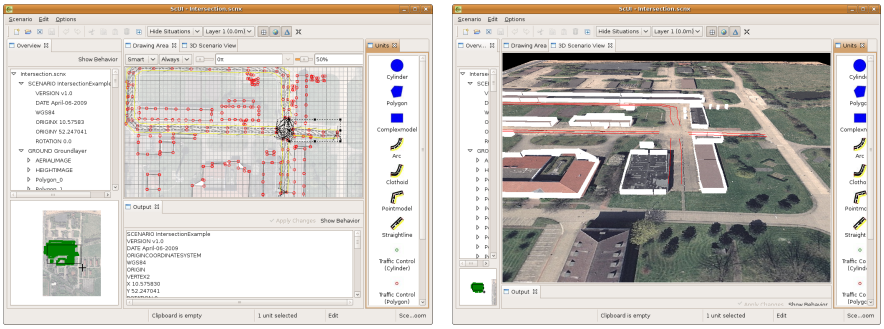


Abbildung 2: Ausschnitt des dynamischen Systemkontextes für eine automotive Anwendung.

Ausschnittsweise wird in den Abbildungen 1 und 2 der statische und dynamische Systemkontext für eine automotive Applikation dargestellt. Dieser Ausschnitt bildet die Grundlage zur Erstellung einer domänenspezifischen Sprache (DSL) zur Modellierung des Systemkontextes. Die Sprache, bestehend aus 36 Entitäten des statischen Systemkontextes sowie 20 Entitäten des dynamischen Systemkontextes basiert sowohl auf Erkenntnissen aus der DARPA Urban Challenge 2007 [RBL+08] als auch aus dem Systemkontext von Autobahnen und Schnellstraßen [WBS+09] und wurde sowohl im Framework MontiCore [GKR+08] zur Verarbeitung durch Java, als auch im Parser-Framework Boost::Spirit [Boo09] zur Verarbeitung durch C++ für die Nutzung in $\mathcal{H}$esperia umgesetzt.

2.2 Textuelle und graphische Modellierung

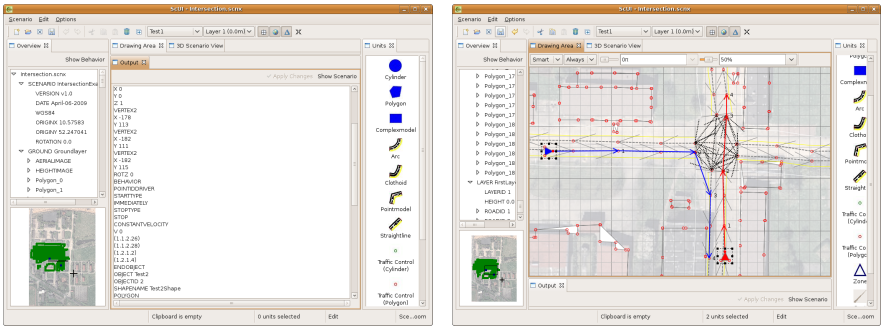
Die Einsatz des Sprachverarbeitungswerkzeugs MontiCore bietet sich an, um einerseits eine prototypische Implementierung der Sprache zu erhalten, diese bereits frühzeitig in der Sprachentwicklung anzupassen und zu verbessern, als auch andererseits die Verwendung der Eclipse Rich Client Platform [The09a] zur Erstellung eines intuitiven graphischen Editors zur Modellierung des statischen sowie dynamischen Systemkontextes zu ermöglichen. In Abbildung 3(a) wird die graphische, zweidimensionale Modellierung des statischen Systemkontextes dargestellt; Abbildung 3(b) veranschaulicht den Systemkontext in einer



(a) Zweidimensionale Modellierung (b) Dreidimensionale Visualisierung

Abbildung 3: Zweidimensionale Szenario-Modellierung (a) und dreidimensionale Visualisierung (b).

vollständig frei navigierbaren, dreidimensionalen Darstellung zur Visualisierung.



(a) Textuelle Modellierung (b) Graphische Modellierung

Abbildung 4: Textuelle (a) und graphische (b) Modellierung einer Verkehrssituation.

Die Definition einer spezifischen Verkehrssituation zur Modellierung des dynamischen Systemkontextes ist in Abbildung 4 in der textuellen (a) sowie graphischen Variante (b) dargestellt. Abgebildet ist eine Kreuzungssituation, in der ein Fahrzeug von Westen kommend seiner blauen Fahrtroute folgend nach Süden abbiegt, und ein zweites Fahrzeug, das von Süden in nördlicher Richtung die Kreuzung überquert. Zur intuitiveren Darstellung der Verkehrsteilnehmer können den Fahrzeugen neben einfachen Zeichnungsprimitiven auch komplexe Modelle gängiger 3D-Modellierungsprogramme zugeordnet werden. Darüber hinaus können für Fahrzeuge ereignisbasierte Abhängigkeiten untereinander definiert werden, die das Wegpunkt-basierte Abfahren einer vorgegebenen Fahrtroute beeinflussen. Hierzu zählen unter anderem die Reaktion auf die Bewegung anderer Teilnehmer oder das Einfahren in spezielle Bereiche zur Auslösung bestimmter Aktionen.

Der modellierte statische Systemkontext wird, neben optionalen Luftbildern und Höhen-

informationen der Umgebung, gemeinsam mit mehreren Situationen zur Beschreibung des dynamischen Systemkontextes als komprimiertes Datenarchiv (*compressed scenario*, SCNX) durch den graphischen Modellierungseditor [Sin08] dem Framework zur Verfügung gestellt.

DSL zur Beschreibung des Systemkontextes

```
1 SCENARIO Braunschweig
2 VERSION v1.0
3 DATE Mai-21-2009
4 ...
5 SURROUNDING
6 SHAPENAME Braunschweig
7 COMPLEXMODEL
8 MODELFILE models/Braunschweig.obj
9 ...
```

Abbildung 5: Auszug einer Instanz des Systemkontextmodells.

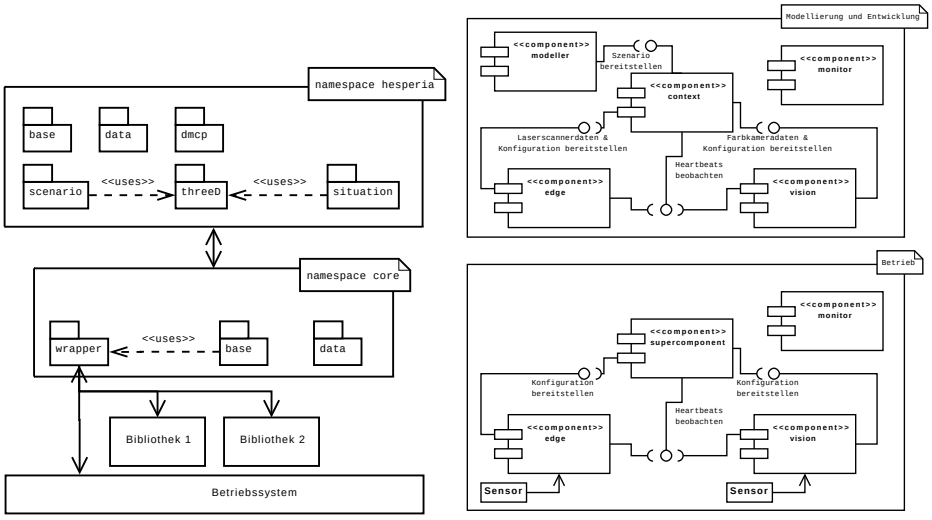
Der Anfang einer konkreten Instanz des Systemkontextmodells ist in Abbildung 5 dargestellt. Es wird mit Meta-Informationen über Version und Erstellungszeitpunkt eingeleitet und bildet dann, je nach gewünschtem Detaillierungsgrad die Elemente auf Abbildung 1 mit den entsprechenden Attributen ab. In Zeile 8 wird bespielsweise auf ein 3D-Modell der Stadt Braunschweig verwiesen, das sich ebenfalls im SCNX-Archiv befindet. Weitere Merkmale des Systemkontextes werden analog definiert.

3 Das Framework $\mathcal{H}$esperia

Zur Nutzung des modellierten Systemkontextes wurde das verteilte, Nachrichten-basierte, zeitgesteuerte Framework $\mathcal{H}$esperia entworfen, das gleichfalls die Entwicklung des Systems selbst erlaubt. Nach eingehenden Analysen der Basis-Software des Versuchsträgers Caroline zur Teilnahme in der DARPA Urban Challenge 2007 [RBL$^+$08, BBB$^+$08] wurde Verbesserungspotential identifiziert und durch eine vollständige Neu-Implementierung behoben, die darüber hinaus Plattform-übergreifend auf Systemen, die entweder mit Microsoft Windows betrieben werden oder POSIX-konform sind, lauffähig ist.

Der weitreichendste Unterschied zwischen beiden Implementierungen ist die Realisierung der grundsätzlichen Kommunikation. Während die Implementierung im Versuchsträger Caroline aufgrund TCP-basierter, stromorientierter 1-zu-1-Kommunikation keine unabhängige, rückkopplungsfreie, nicht-blockierende Kommunikation zuließ und die Datenmenge zudem linear mit der Anzahl verwendeter Prozesse direkt proportional zunahm, erlaubt $\mathcal{H}$esperia durch Nutzung von UDP-basiertem Multicast als grundsätzliche Kommunikation skalierungsfreien Nachrichtenaustausch zwischen mehreren verteilten Prozessen.

In Abbildung 6(a) ist die Paketübersicht dargestellt. Grundsätzlich teilt sich das Framework in zwei Pakete `core` und `hesperia` auf. Das Paket `core` stellt einerseits die Plattform-Unabhängigkeit sicher, indem alle verwendeten Bibliotheken wie etwa Boost::-



(a) Paketübersicht

(b) Komponentendiagramm während der Entwicklung und im Betrieb

Abbildung 6: (a) zeigt die Paketübersicht, (b) stellt notwendige Komponenten während der Entwicklung und für den Betrieb dar.

Spirit [Boo09], Realtime-Erweiterungen und Systemaufrufe ausschließlich durch Adapter an höhere Schichten zur Verfügung gestellt werden. Damit wird sowohl die Bedienung vereinfacht als auch zukünftige Austauschbarkeit sichergestellt. Das Paket `hesperia` selbst stellt Konzepte der durch `core` angebotenen Funktionalität zur Verfügung. Hier wird beispielsweise eine sogenannte `ClientConference` realisiert, die automatisch den Empfang von per UDP-Multicast versandten Nachrichten über Datentyp-spezifische Observer, wie auch das Versenden selbst transparent darauf aufbauenden Komponenten zur Verfügung stellt. Im Kern des Pakets `hesperia` steht jedoch die Integration der domänenspezifischen Sprache zur Modellierung des Systemkontextes sowie deren Transformation in einen 3D-Szene-Graphen zur Generierung von Sensorrohdaten (siehe Abschnitt 4).

Darüber hinaus sind in Abbildung 6(b) zwei Komponentendiagramme dargestellt, in denen die Nutzung der durch $\mathcal{H}$esperia realisierten Komponenten skizziert wird. Zur Modellierung und Entwicklung nutzen die *components-under-development* (CUDs), hier durch die Komponenten `edge` und `vision` angedeutet, den `context`, um Sensorrohdaten zu erhalten. Der `context` stellt die Komponente zur Generierung der für die CUDs notwendigen Eingabedaten dar, indem die benötigten Informationen direkt aus der Beschreibung des Systemkontextes, die durch die graphische Modellierung bereitgestellt wird, abgeleitet werden. Darüber hinaus steuert `context` die für alle Komponenten gültige Systemzeit. Im realen Betrieb hingegen wird der `context` durch eine überwachende `supercomponent` ersetzt, die per *dynamic module configuration protocol* (DMCP) die laufenden Komponenten zentral konfiguriert und überwacht. In beiden Konfigurationen

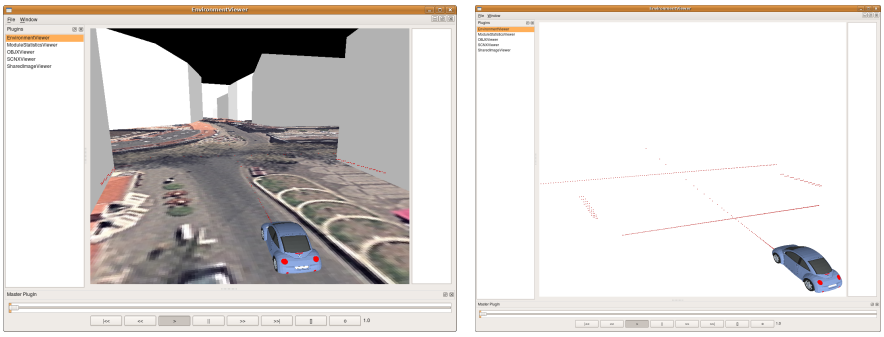
kann das laufende System rückwirkungsfrei durch beliebig viele Instanzen von `monitor` beobachtet und für spätere Analysen aufgezeichnet werden.

4 Virtuelle Sensoren

Wie bereits in der Einleitung erwähnt, ermöglicht das im vorangegangenem Abschnitt beschriebene Framework unter Nutzung des durch die DSL beschriebenen Systemkontextes die Generierung von Sensorrohdaten zur Nutzung im Rahmen des *virtual prototypings* in der Software-Entwicklung. Hierzu wurde das in [BR09] vorgestellte Verfahren weiterentwickelt und in $\mathcal{H}$esperia integriert.

4.1 Generierung von Daten für 1-Ebenen-Laserscanner

Grundlage des Verfahrens ist eine in OpenGL renderbaren Szene zur Plattform-übergreifenden Nutzung einer *graphical processing unit* (GPU). Hierfür wird durch die Komponente `context` aus $\mathcal{H}$esperia der modellierte Systemkontext zur Initialisierung eingelesen und das durch einen Adapter gekapselte Boost::Spirit zur Erzeugung einer Instanz des abstrakten Syntaxbaumes des Systemkontextmodells genutzt. Der Baum wird anschließend durch einen Visitor traversiert und der formal beschriebene statische sowie dynamische Systemkontext durch Zeichnungsprimitive aus OpenGL als Szene-Graph abgeleitet. Dieser Szene-Graph wird dann für das eigentliche Rendering verwendet.



(a) Szene mit drei Scanebenen (b) Ausblendung statischer Objekte

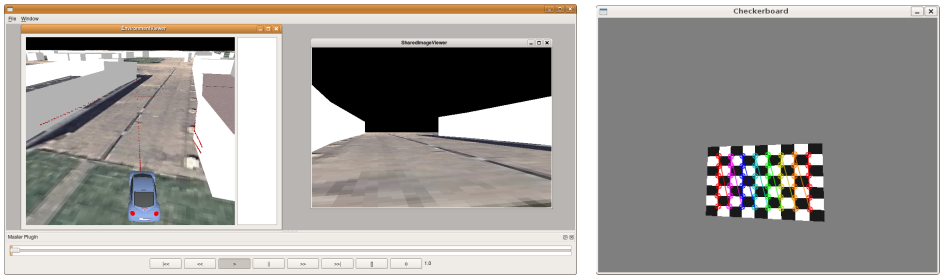
Abbildung 7: Einsatz des modellierten Systemkontextes zur Simulation von drei 1-Ebenen-Laserscannern. (a) zeigt die Szene inklusive Messergebnisse, (b) stellt nur die Messergebnisse dar.

In Abbildung 7 wird ein modellierter Systemkontext zur Generierung von Sensorrohdaten dargestellt. In (a) ist neben einem Luftbild in der Auflösung von $0{,}1\frac{m}{px} * 0{,}09\frac{m}{px}$ maßstabsgetreu ein 3D-Modell der Innenstadt von Braunschweig in den Ausmaßen von etwa $1.400m * 1.400m$, bestehend aus nicht-texturierten 44.436 Einzeldreiecken, abgebildet.

In dieser Szene bewegt sich ein Sensoren-Träger, der mit drei 1-Ebenen-Laserscannern ausgerüstet ist. Zwei Sensoren liefern Distanzdaten orthogonal zur Fahrtrichtung aus 8m und 25m Entfernung vor dem Fahrzeug, ein dritter Sensor ist parallel zur Fahrtrichtung orientiert und liefert Daten ab 0,5m vor dem Fahrzeug.

Wie in [BR09] bereits dargestellt, liegen die Messpunkte in absoluten Koordinaten vor. Zur Nutzung der Daten im Rahmen der Algorithmenentwicklung zur Sensorrohdatenverarbeitung für Sick LMS 291-Laserscanner [Sic09], werden die einzelnen Messpunkte unter Berücksichtigung der aktuellen absoluten Position und Rotation des Sensoren-Trägers sowie der virtuellen Montagepositionen der Sensoren in das lokale Sensorkoordinatensystem transformiert. Basierend auf der Datenspezifikation des Herstellers werden diese Daten anschließend nachgelagerten Komponenten zur Verfügung gestellt.

4.2 Generierung von Daten für eine Farbkamera



(a) Monitoring-Anwendung zur Visualisierung der Umgebungsdaten und Darstellung des Kamerabildes

(b) Kalibrierkörper zur Bestimmung intrinsischer Kameraparameter

Abbildung 8: (a) Monitoring-Applikation zur Visualisierung von angereicherten Umgebungsdaten (linkes Teilfenster) und einer virtuellen Kamera (rechtes Teilfenster). (b) Automatische Bestimmung intrinsischer Kameraparameter.

Über die Generierung von Rohdaten für 1-Ebenen-Laserscanner bietet es sich offenkundig darüber hinaus an, direkt Daten für virtuelle Kameras aus der 3D-Szene abzuleiten. Wie in Abbildung 8 dargestellt, wird direkt das derzeitige Kamerabild in einen *shared memory*-Speicherbereich zur Nutzung innerhalb eines Bildverarbeitungsalgorithmus oder für die Darstellung in der Komponente `monitor` bereitgestellt. Darüber hinaus ermöglicht $\mathcal{H}$esperia, wie in Abbildung 8(b) gezeigt, auch direkt die Bestimmung der extrinischen und intrinischen Kameraparameter durch die Verwendung spezieller Kalibrierkörper.

4.3 Leistungsfähigkeit des Frameworks $\mathcal{H}$esperia

In Abbildung 9 ist die Simulationsleistung zur Generierung der Sensorrohdaten für unterschiedliche Systemkontexte dargestellt. Die Berechnungen wurden auf einem Intel® Core™ 2 Duo T9600 mit 2,8GHz mit einer NVIDIA Quadro NVS 160M mit 512MB Videospeicher und 4GB Arbeitsspeicher durchgeführt.

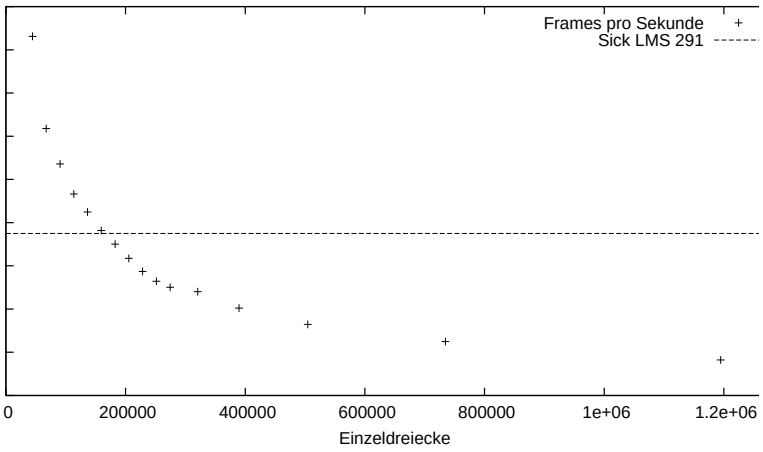


Abbildung 9: Simulationsleistung des $\mathcal{H}$esperia-Frameworks: + bezeichnet die Messreihe zur Datengenerierung für drei 1-Ebenen-Laserscanner bei unterschiedlich komplexen Szenarien; mit - ist zur Referenz die Leistung eines Sick LMS 291 Laserscanners aufgetragen.

Die mit + bezeichnete Messreihe stellt die berechneten Frames pro Sekunde für unterschiedliche Systemkontexte dar. Das bereits erwähnte Modell der Innenstadt von Braunschweig setzt sich aus 44.436 Einzeldreiecken zusammen. Die Leistung beträgt hier 166,24 Frames pro Sekunde. Damit liegt die für den im Beispiel modellierten Systemkontext erzielte Simulationsleistung ca. 130% über der Frequenz realer Sensoren. Bei gleichzeitiger Generierung von Farbkameradaten liegt die Leistung bei etwa 117% über den realen Sensoren.

Um die Komplexität des modellierten Systemkontextes zu erhöhen, wurden zusätzlich zum Modell der Innenstadt von Braunschweig noch Höheninformationen in der Elevationsauflösung von 6mm für einen Bereich von $1.400m * 1.400m$ benutzt. Das Höhenfeld wurde in Form eines Gitternetzes abgebildet, was zu einer Komplexität von insgesamt 1.194.786 Einzeldreiecken führte und eine Leistungsfähigkeit von 16,48 Frames pro Sekunde ergab.

5 Verwandte Arbeiten

Dieser Beitrag ergänzt die Vorarbeiten aus [BBR07] und [BBR08] maßgeblich um den wichtigen Bestandteil der Generierung von komplexen, gezeiteten Eingabedaten. Die Vorarbeiten beschäftigten sich mit der Generierung synthetischer Daten für Algorithmen mit abstrakten und bereits durch eine Sensordatenvorverarbeitung aggregierten Daten. Dieser Beitrag vervollständigt das *virtual prototyping* durch Einbeziehung der gesamten Verarbeitungskette des Systems unter Verwendung einer GPU.

Einen ebenfalls auf einer GPU basierenden Ansatz zur Generierung von Sensorrohdaten verwendet das CyCAB von INRIA [INR08]. Im Gegensatz zum Beitrag [BR09] muss hierbei allerdings eine vollständige Analyse über das Tiefenwertbild der aktuellen Szene durchgeführt werden. Einen vergleichbaren Ansatz setzte das Team der Freien Universität Berlin für ihren Beitrag in der DARPA Urban Challenge 2007 ein [RRG+07].

Die Simulationsumgebung CarMaker [IPG09] der Firma IPG erlaubt einen ähnlichen Ansatz. Originär liegt der Fokus bei CarMaker in der Fahrdynamiksimulation. Erweiterungen dazu unterstützen die Entwicklung durch die Generierung von Sensordaten. Im Gegensatz zu diesem Beitrag stützt sich die Software allerdings ausschließlich auf die Erzeugung von Sensordaten von im Vorfeld definierten Objektkonturen. Darüber hinaus reichende Eingabedaten, die durch andere Elemente des Systemkontextes wie Bäume oder Bordsteinkanten verursacht werden, werden hierbei nicht betrachtet.

Neben dem hier dargestellten Framework $\mathcal{H}$esperia ist das *Automotive Data and Time-triggered Framework (ADTF)* zu benennen, das eine einheitliche Programmierschnittstelle zur Software Entwicklung anbietet [Sch07]. Die Aktualität und Praxisrelevanz unserer Vorarbeiten [BBR07] und [BBR08] und dieses Beitrags werden dadurch unterstrichen, dass derzeit das ADTF selbst um Elemente für virtuelle Testfahrten erweitert wird [vNCDW09].

Aus dem Roboterfußball entstammen die Anwendungen Player/Stage/Gazebo [GVH03]. Der Fokus liegt hier auf der Vereinfachung der Software-Entwicklung insbesondere für diesen Wettbewerb [Pad09]. Das Framework fokussiert dabei auf spezielle Hardware-Plattformen, die für automotive Software-Entwicklung aufgrund Unterschiede in ihrer Aktorik und insbesondere ihrer Regelung eher ungeeignet erscheinen. Darüber hinaus basiert die Generierung der Sensorrohdaten innerhalb des Systems auf dem Prinzip des Raytracings, was sich insbesondere bei komplexen Szenen für interaktives *virtual prototyping* als zu berechnungsintensiv erweist.

Das Nachfolgeprojekt “Stadtpilot” zum Projekt CarOLO an der Technischen Universität Braunschweig erwägt ebenfalls die Nutzung eines entwicklungsbegleitenden Simulations- und Testkonzepts [SMWM09]. Der ausschließliche Schwerpunkt liegt hier in der reproduzierbaren Erzeugung möglichst vieler verschiedener Trajektorien zur Qualitätssicherung der Regelalgorithmen. Damit wird allerdings das Problem der Generierung von Daten für die virtuelle Erprobung von Sensorfusionsalgorithmen nicht betrachtet.

Einen zum CarMaker vergleichbaren Ansatz stellte die Firma IAV auf der diesjährigen AAET vor [SZ09]. Deren Vorgehen deckt sich insofern mit dem Ansatz der Firma IPG, da auch explizit zu detektierende Umfeldobjekte modelliert werden müssen. Anschließend wird der jeweils aktuelle Sichtbereich des Sensors als Polygon modelliert und mit den

Objekten der Umgebung geschnitten. Die daraus resultierenden Konturdaten werden zur Weiterverarbeitung bereitgestellt. Dieses Vorgehen erlaubt ebenfalls nicht die Verarbeitung von weiteren, nicht explizit modellierten Elementen der Umgebung, wie etwa Bordsteinkanten, so dass die Entwicklung einer Sensordatenfusion nur mit stark reduzierten, idealisierten Umgebungsdaten erprobt werden kann.

6 Zusammenfassung

In diesem Beitrag wurde mit $\mathcal{H}$esperia ein Komponenten-basiertes Framework zur Szenario-gestützten Modellierung und Entwicklung Sensor-basierter Systeme vorgestellt, das prototypisch in der Programmiersprache C++ Plattform-übergreifend realisiert wurde. Innerhalb dieses Frameworks wurde eine Komponente zur Generierung von Sensorrohdaten sowohl für 1-Ebenen-Laserscanner als auch für Farbkameras realisiert, die zur Generierung der Daten sowohl die CPU als auch moderne Graphikprozessoren (GPU) einsetzen. Grundlage für die Generierung von Sensorrohdaten ist hierbei die Modellierung des gewählten Systemkontextes. Zur Modellierung wurde eine domänenspezifische Sprache entworfen, mittels der sich beispielsweise der Kontext eines Automobils, gegliedert nach statischen und dynamischen Elementen, über einen graphischen Editor intuitiv modellieren lässt.

Wie experimentell für unterschiedliche Systemkontexte für die Entwicklung von Sensor-basierter Umfeldwahrnehmung gezeigt wurde, liegt die Simulationsleistung über der Frequenz realer Sensoren für Systemkontexte, die beispielsweise eine Innenstadt umfassen. Damit erweist sich das Framework zur Szenario-gestützten Modellierung und Entwicklung Sensor-basierter Systeme als geeignet, bereits frühzeitig im Software-Entwicklungsprozess eingesetzt zu werden, um komplexe und zeitaufwändige Situationen zunächst gefahrlos in Laborversuchen zu erproben, bevor das eingebettete System auf einem realen Sensoren-Träger in Betrieb genommen und getestet wird.

Literatur

[BBB+08] Christian Basarke, Christian Berger, Kai Berger, Karsten Cornelsen, Michael Doering, Jan Effertz, Thomas Form, Tim Gülke, Fabian Graefe, Peter Hecker, Kai Homeier, Felix Klose, Christian Lipski, Marcus Magnor, Johannes Morgenroth, Tobias Nothdurft, Sebastian Ohl, Fred W. Rauskolb, Bernhard Rumpe, Walter Schumacher, Jörn-Marten Wille, and Lars Wolf. Team CarOLO – Technical Paper. Informatik-Bericht 2008-07, Technische Universität Braunschweig, October 2008.

[BBKR09] Arne Bartels, Christian Berger, Holger Krahn, and Bernhard Rumpe. Qualitätsgesicherte Fahrentscheidungsunterstützung für automatisches Fahren auf Schnellstrassen und Autobahnen. In Gesamtzentrum für Verkehr Braunschweig e.V., editor, *AAET 2009 – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, volume 10, pages 341–353, February 2009.

[BBR07] Christian Basarke, Christian Berger, and Bernhard Rumpe. Software & Systems Engi-

neering Process and Tools for the Development of Autonomous Driving Intelligence. *Journal of Aerospace Computing, Information, and Communication*, 4(12):1158–1174, December 2007.

[BBR08] Christian Basarke, Christian Berger, and Bernhard Rumpe. Using Intelligent Simulations for Quality Assurance in the 2007 DARPA Urban Challenge. Workshop Contribution, May 2008. Workshop-Beitrag.

[Boo09] Boost. Boost::Spirit. http://spirit.sourceforge.net/, April 2009.

[BR09] Christian Berger and Bernhard Rumpe. Nutzung von projektiven Texturen auf einer GPU zur Distanzmessung für automotive Sensorsimulationen. In Gesamtzentrum für Verkehr Braunschweig e.V., editor, *AAET 2009 – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, volume 10, pages 319–329, February 2009.

[Bro05] Manfred Broy. Automotive Software and Systems Engineering. In *Proceedings of IEEE International Conference on Formal Methods and Models for Co-Design*, pages 143–149, 2005.

[GKR+08] Hans Grönniger, Holger Krahn, Bernhard Rumpe, Martin Schindler, and Steven Völkel. MontiCore: a framework for the development of textual domain specific languages. In *ICSE Companion '08: Companion of the 30th international conference on Software engineering*, pages 925–926, New York, NY, USA, 2008. ACM.

[GVH03] Brian Gerkey, Richard T. Vaughan, and Andrew Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[INR08] INRIA. Bayesian Approach to Cognitive Systems. Technical report, INRIA, 2008.

[IPG09] IPG. CarMaker. http://www.ipg.com/, April 2009.

[Kan95] Cem Kaner. Software Negligence and Testing Coverage. *Software QA Quarterly*, 2(2):18, 1995.

[Pad09] PaderKicker. PaderKicker Fuballroboter. http://paderkicker.de/p/paderkicker/, April 2009.

[RBL+08] Fred W. Rauskolb, Kai Berger, Christian Lipski, Marcus Magnor, Karsten Cornelsen, Jan Effertz, Thomas Form, Fabian Graefe, Sebastian Ohl, Walter Schumacher, Jörn-Marten Wille, Peter Hecker, Tobias Nothdurft, Michael Doering, Kai Homeier, Johannes Morgenroth, Lars Wolf, Christian Basarke, Christian Berger, Tim Gülke, Felix Klose, and Bernhard Rumpe. Caroline: An Autonomously Driving Vehicle for Urban Environments. *Journal of Field Robotics*, 25(9):674–724, September 2008.

[RRG+07] Javier Rojo, Raúl Rojas, Ketill Gunnarsson, Mark Simon, Fabian Wiesel, Fabian Ruff, Lars Wolter, Frederik Zilly, Neven Santrač, Tinosch Ganjineh, Arash Sarkohi, Fritz Ulbrich, David Latotzky, Benjamin Jankovic, Gretta Hohl, Thomas Wisspeintner, Stefan May, Kai Pervölz, Walter Nowak, Francesco Maurelli, and David Dröschel. Spirit of Berlin: An Autonomous Car for the DARPA Urban Challenge – Hardware and Software Architecture. Technical report, Freie Universität Berlin, June 2007.

[Sch07] Roland Schabenberger. ADTF: Framework for Driver Assistance and Safety Systems. In VDI Wissensforum IWB GmbH, editor, *Integrierte Sicherheit und Fahrerassistenzsysteme*, number 2000, pages 701–710. VDI-Gesellschaft Fahrzeug- und Verkehrstechnik, October 2007.

[Sic09] Sick. Lasermesssysteme LMS200/LMS211/LMS221/LMS291. http://mysick.com/, April 2009.

[Sin08] Niels Sintara. Entwurf und Realisierung eines plattformübergreifenden, graphischen Verkehrsszenario-Editors. Master's thesis, Technische Universität Braunschweig, December 2008.

[SMWM09] Falko Saust, Tobias Müller, Jörn Marten Wille, and Markus Maurer. Entwicklungsbegleitendes Simulations- und Testkonzept fr autonome Fahrzeuge in stdtischen Umgebungen. In Gesamtzentrum für Verkehr Braunschweig e.V., editor, *AAET 2009 – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, volume 10, pages 87–107, February 2009.

[SZ09] Benedikt Schonlau and Rene Zschoppe. Test und Absicherung von Funktionen mit synthetischen Umfeld- und Fahrzeugeigendaten. In Gesamtzentrum für Verkehr Braunschweig e.V., editor, *AAET 2009 – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, volume 10, pages 106–121, February 2009.

[The09a] The Eclipse Foundation. Eclipse Rich Client Platform. http://www.eclipse.org/rcp/, April 2009.

[The09b] The Eclipse Foundation. Eclipse Test & Performance Tools Platform Project. http://www.eclipse.org/tptp/, April 2009.

[Tig09] Tigris.org. CxxTest. http://cxxtest.tigris.org/, April 2009.

[Und07] Mark Underseth. The complexity crisis in embedded software. *Embedded Computing Design*, pages 31–33, April 2007.

[vNCDW09] Kilian von Neumann-Cosel, Marius Dupuis, and Christian Weiss. Virtual Test Drive - Provision of a Consistent Tool-Set for [D,H,S,V]-in-the-Loop. In *Proceedings on Driving Simulation Conference*, Februar 2009.

[WBS+09] Andreas Weiser, Arne Bartels, Simon Steinmeyer, Karsten Schultze, Marek Musial, and Kristian Weiß. Intelligent Car – Teilautomatisches Fahren auf der Autobahn. In Gesamtzentrum für Verkehr Braunschweig e.V., editor, *AAET 2009 – Automatisierungssysteme, Assistenzsysteme und eingebettete Systeme für Transportmittel*, volume 10, pages 11–26, February 2009.