HLA-basierte Kosimulation domänentypischer Simulatoren*

Stefan Puch, Gerald Sauter und Martin Fränzle

Carl von Ossietzky Universität, Oldenburg, Germany
{fraenzle|puch|sauter}@informatik.uni-oldenburg.de

1 Einleitung

Mit der zunehmenden Verbreitung von Fahrerassistenzsystemen, welche die Fahrsituation bewerten und entweder über Rückmeldung an den Fahrzeugführenden (beispielsweise Lane Assist, Side Assist) oder durch unmittelbare fahrdynamische Stelleingriffe (z.B. ESP, City Safety) zu beeinflussen suchen, stellt sich vermehrt die Frage der situativen Angemessenheit dieser Einflussnahme. Im Extremfall erreicht die Assistenz hier exakt das Gegenteil des intendierten Effekts, indem sie vom Fahrzeugführenden in Situationen ohnehin hoher Belastung zusätzliche Aufmerksamkeit beansprucht, ihn mit unerwarteten Eingriffen oder gar unbemerktem Übergang in andere Betriebsmodi überrascht, die Kontrolle in für den Normalfahrzeugführenden unkontrollierbaren fahrdynamischen Zuständen an den Fahrzeugführenden zurücküberweist, oder durch Überlagerung von Stellmomenten den fahrdynamischen Regelkreis zwischen Fahrzeug und -führendem destabilisiert. Derartige Effekte sind selbstverständlich nicht völlig vermeidbar, ihre Häufigkeit — genauer: ihre Häufigkeit in kritischen Situationen, in welchen die vorgenannten Einflüsse ursächlich für Unfälle werden können — muss jedoch so weit reduziert werden, dass insgesamt ein deutlicher, dem Kostenaufwand für das Assistenzsystem angemessener, Sicherheitsgewinn erzielt wird.

Aus dem Vorgenannten folgt, dass eine umfassende Bewertung des Sicherheitseinflusses geplanter Assistenz bereits in deren frühen Entwurfsphasen wünschenswert ist. Hierbei sind naturgemäß alle möglichen Einsatzszenarien des offenen Systems "Auto" zu berücksichtigen, im Idealfall mit zutreffenden Auftretenswahrscheinlichkeiten. Eine derartige Überdeckung des Verhaltensraums des gekoppelten Systems Mensch-Assistenzsystem-Fahrzeug-Verkehrsumgebung wird durch Realfahrten aber erst in jahrelangem Einsatz, verbunden mit entsprechender Gewöhnung an die Assistenz, erreicht. In der Entwurfsphase muss deshalb auf simulierte Fahrten zurückgegriffen werden, welche ein Provozieren spezifischer Fahrsituationen bei vertretbarem Risiko gestatten. Wegen des hohen Zeit- und Kostenaufwands kann aber auch hierbei, trotz Verfügbarkeit verschiedener Testfahrzeuge und Simulatoren und damit unterschiedlicher Kompromisse zwischen Realismus der Situationsdarstellung und -analyse einerseits und Aufwand andererseits (vgl. Abb. 1), dennoch nur eine vergleichsweise geringe Überdeckung des immensen Fahrszenarienraums erlangt werden. Im IMoST-Projekt wird deshalb eine Ergänzung der Sicherheitsanalyse durch modellbasierte Analysetechniken vorgeschlagen, welche wegen der direkten Verfügbarkeit am Arbeitsplatz des Assistenzsystementwickelnden nicht nur eine unmittelbare experimentelle Überprüfung jeglicher Entwurfsentscheidung und damit noch frühere Abwendung von Entwurfsfehlern, sondern auch eine wesentlich bessere Überdeckung des Fahrszenarienraums gestatten. Dies gilt insbesondere im Falle einer vollständigen Automatisierung der modellbasierten Analyse, welche IMoST zur Verfügung stellen möchte. Dem Verlust an Genauigkeit durch die — im Falle eines Mensch-Maschine-Systems notwendigerweise idealisierende und auf Annahmen über menschliches Verhalten beruhende — Modellbildung tritt dabei ein Genauigkeitsgewinn durch um Größenordnungen bessere Überdeckung des Verhaltensraums, insbesondere des Teilraums riskanter, seltener oder schwer provozierbarer Situationen, gegenüber. Damit weisen experimentbasierte und modellbasierte Sicherheitsanalysen zu einander komplementäre Stärken und Schwächen auf (vgl. Abb. 1), was ihren gemeinsamen Einsatz in einem Assistenzsystementwurfsprozess attraktiv macht.

Eine modellbasierte Analyse des fahrdynamischen Verhaltens eines mit Assistenz ausgestatten Kraftfahrzeugs bedarf der folgenden drei Komponenten:

* This work has been partially supported by the Ministry for Science and Culture of Lower Saxony as part of the interdisciplinary research project "Integrated Modelling for Safe Transportation" (IMoST).

Abbildung 1. Verschiedene Formen der Assistenzsystemerprobung: Versuchsfahrt, simulierte Versuchsfahrt im beweglichen bzw. starren Fahrsimulator, modellbasierte simulierte Versuchsfahrt (v.l.n.r.)

1. einer Modellbildung der Dynamik des Gesamtsystems aus Mensch, Assistenzsystem, Fahrzeug und Verkehrsumgebung,
2. Methoden zur Generierung von Fahrtrajektorien (oder, obwohl technisch verschieden, im Nutzen weitgehend äquivalent: erreichbarer Zustandsräume) des Gesamtmodells,
3. Algorithmen zur Exploration und Analyse dieses Verhaltensraums, insbesondere zur Gewinnung von Häufigkeitsstatistiken.

Im vorliegenden Bericht befassen wir uns mit Punkt 2 der vorgenannten Anforderungen. Hierzu stünden als grundlegende Techniken prinzipiell entweder zustandsexplorative Verifikationsverfahren oder Simulation zur Verfügung. Leider ist in absehbarer Zeit nicht mit hinreichender Leistungsfähigkeit zustandsexplorativer Verifikationsverfahren zu rechnen: Zum einen sind diese bislang nicht für sämtliche Komponenten des Mensch-Maschine-Systems und die dabei naturgemäß verwendeten heterogenen Modellatome verfügbar, sondern lediglich für die Klasse nicht-linearer, probabilistischer hybrid diskret-kontinuierlicher Systeme als größter Teilklasse [4], zum Anderen endet ihre Skalierbarkeit bei viel zu geringen Systemdimensionen. Aktuell sind hybride Systeme bei linearen Differentialgleichungen mit einigen Hundert und bei nicht-lineare nmit einigen Zehner Dimensionen behandelbar, wenn das System deterministisch und damit frei von Verzweigungen nicht-deterministischer oder probabilistischer Art ist. Leider sind die in IMoST betrachteten Systeme natürlicherweise offen und weisen mithin Verzweigungen auf, was die handhabbare Systemgröße um etwa eine weitere Größenordnung reduziert. Aus diesem Grunde kommen im Kontext der Assistenzsystementwicklung derzeit nur simulationsbasierte Verfahren zur Berechnung des dynamischen Verhaltens des Gesamtmodells in Frage. Ein solches Simulationsverfahren muss die Komponenten Mensch, Assistenzsystem, Fahrzeug und Verkehrsumgebung ko-simulieren, um ihr gekoppeltes dynamisches Verhalten zu erzeugen. Um eine Verhaltensanalyse durch Exploration im Sinne von Punkt 3 der oben genannten Anforderungen zu ermöglichen, muss bei dieser gekoppelten Simulation auf zeit- und zustandstreue an den Schnittstellen zwischen den Komponenten geachtet werden, da ihre gemeinsame Dynamik in der Realwelt über das für alle Komponenten gemeinsame Fortschreiten der Weltzeit[1] synchronisiert wird.

Der Bericht beschäftigt sich insbesondere mit der technischen Realisierung einer zeitgetreuen Kosimulation von Mensch, Assistenzsystem, Fahrzeug und Verkehrsumgebung. In Abschnitt 2 wird dazu auf verschiedene Formen der Ko-Simulation heterogener Komponenten eingegangen. Dabei wird die Konföderation von Off-the-Shelf-Simulatoren als einzig praktikabler Weg zur Realisierung einer Ko-Simulation von Mensch, Assistenzsystem, Fahrzeug und Verkehrsumgebung unter den

[1] Relativistische Effekte spielen bei den vorliegenden relativgeschwindigkeiten und Entfernungen zwischen den Verkehrsteilnehmern keine Rolle, so dass von einer gemeinsamen globalen Zeit ausgegangen werden darf.

Projektbedingungen identifiziert. Abschnitt 3 beschreibt dann die High-Level-Archtitecture (HLA, IEEE Standard 1516) als Basistechnologie zur Erlangung von temporaler und Zustandkonsistenz in einer Ko-Simulation aus heterogenen Off-the-Shelf-Simulatoren. Im Abschnitt 4 schließlich wird detailliert die Umsetzung einer derartigen Ko-Simulation auf HLA-Basis beschrieben.

2 Formen der Ko-Simulation heterogener Komponenten

Für die Ko-Simulation heterogener Systeme, wie dem hier vorliegenden Mensch-Maschine-System, gibt es im Wesentlichen vier Grundkonzepte:

1. Händische Erzeugung eines aufgabenspezifischen Simulationsprogramms, welches sowohl die Fortschreibung der internen Zustände der einzelnen Systemkomponenten als auch die Synchronisation dieser Zustandupdates und deren wechselweises Sichtbarwerden an den Komponentenschnittstellen explizit durch seinen Kontrollfluss moderiert.
2. Manuelle Einbettung der Komponentenmodelle in eine gemeinsame Modellierungsprache (bspw. Modelica oder Simulink [7]) und deren Ausführung in der zugehörigen Simulationsplattform.
3. Erstellung und Pflege der einzelnen Komponentenmodelle in jeweils domänenspezifischen Modellierungsprachen, gefolgt von automatischer Übersetzung in eine gemeinsame Metasprache und Ausführung in der zugehörigen Simulationsplattform (z.B. verfolgt im Projekt SPEEDS [8]).
4. Kopplung von domänenspezifischen Simulatoren über Nachrichtenaustausch zwecks wechselweisen Updates der Schnittstellenzustände sowie ggf. — und idealerweise — Synchronisation der lokalen Fortschritte der Simulationszeit.

Aus den Umständen des IMoST-Projektes ergibt sich unmittelbar, dass die Ansätze 1 bis 3 im Projektkontext ungeeignet sind, da sie zentrale Anforderungen nicht erfüllen:

- Die Komponentenmodelle sind nicht fix, sondern werden fortlaufend in Kooperation mit anderen Projekten weiterentwickelt, wobei domänentypische Modellierungssprachen (bspw. Regelsysteme für die Modelle kognitiver Fähigkeiten des Fahrzeugführenden, Simulink-Stateflow-Modelle für das Assistenzsystem) Verwendung finden. Ansätze der Formen 1 und 2, welche eine Formulierung sämtlicher Teilmodelle in einer gemeinsamen Programmier- oder Modellierungsprache erfordern, unterstützen die erforderliche Heterogenität des Modellierungstils nicht.
- Für einige Komponenten ändert sich die natürliche Modellierungsform entlang der Entwurfskette, was Ko-Simulation der Form 1 erschwert. Dies ist beispielsweise für das Assistenzsystem der Fall, bei dem von einem idealen Reglermodell sukzessive zu einer Softwarearchitektur übergegangen wird.
- Einige Komponenten besitzen extrem umfangreiche vorgefertigte “Black-Box”-Modelle (bspw. die Verkehrssimulation), deren Nachbildung in allgemeinen Modellierungsprachen unverhältnismäßig aufwändig wäre, so dass entsprechende “Off-the-Shelf”-Simulatoren eingebunden werden können, was mit den Konzepten 2 und 3 kaum möglich ist.

Aus diesen Gründen erweisen sich die ersten drei der vorgenannten Kosimulationsansätze als ungeeignet. Eine Kosimulation in Form einer domänentypischer Off-the-Shelf-Simulatoren mit einem einheitlichen Interface wäre jedoch ein geeigneter Lösungsansatz, wenn sie die für die Analyse notwendige temporale und Zustansdkonsistenz der Komponentensimulatoren sicherstellt.

Der interdisziplinäre Forschungsverbund IMoST [2] verfolgt exakt diesen Ansatz. Er entwickelt unter Zuhilfenahme geeigneter Kosimulationsinfrastruktur einen Kosimulator für ein Modell, welches aus einem Fahrermodell, einem Assistenzsystemmodell sowie einem handelsüblichen Fahrsimulator inklusive Verkehrssimulation besteht. Zur Erfüllung der gesetzten Ziele standen dabei drei verschiedene Komponentensimulatoren zur Verfügung:

1. Der Fahrschulsimulator ST-Software, der von einer Versuchsperson bedient werden kann und der neben dem Eigenfahrzeug mit dessen Fahrphysik auch die Umgebung einschließlich weiterer Verkehrsteilnehmer simuliert,

2. die Entwicklungsumgebung MATLAB/ Simulink/ Stateflow, die in der Industrie weit verbreitet ist und sich zum modellbasierten Entwurf eingebetteter HW-SW-Systeme hervorragend eignet,
3. sowie ein Werkzeug für die Modellierung und Simulation menschlichen Operatorverhaltens, welches im Folgenden als "kognitives Modell" bezeichnet wird.

Jeder der drei Simulatoren erfüllt die Anforderungen des jeweiligen Bereichs bezüglich geeigneter Modellbildung. Wegen der unterschiedlichen Domänen sind die Modellierungsparadigmen dabei notwendigerweise heterogen. So verwendet das kognitive Modell beispielsweise einen regelbasierten Ansatz, um die menschlichen Gedächtnisprozesse nachzubilden. In MATLAB/ Simulink/ Stateflow können hingegen Datenflussnetzwerke und eingebettete Automaten über diskreter oder kontinuierlicher Zeit modelliert werden. Um nun die speziellen Eigenschaften der einzelnen Simulatoren bei einer Kopplung zu erhalten und auch ihr intern verwendetes Zeitmodell zu bewahren, wurden die Simulatoren mittels einer geeigneten Broker-Architektur zu einer Kosimulation zusammengeschlossen. Hierbei fiel die Wahl auf die High-Level-Architecture (HLA), da diese einen leistungsfähige und bewährte Standard darstellt.

3 Die High Level Architecture

Die HLA ist in dem IEEE-Standard 1516-2000 [13,14,15] spezifiziert und bietet die Möglichkeit, verschiedene Simulatoren, Federates genannt, in einer Gesamtsimulation, Federation genannt, zu vereinigen. Hierbei übernimmt die HLA die Weiterleitung von lokalen Zustandsupdates an alle anderen Komponenten, in deren Schnittstelle dieser Update sichtbar ist, sowie die Synchronisation des globalen Zeitfortschritts. Um das erstere, also die Weiterleitung von Zustandänderungen, zu gewährleisten, können Simulatorkomponenten bei der HLA Zustandsvariablen von anderen Komponenten "abonnieren". Die HLA sorgt dann dafür, dass alle Updates der variablen an alle Abonnenten weitergeleitet werden. Bei der Zeitsynchronisation ist das Grundprinzip, dass jede Simulatorkomponente gewünschten Fortschritt der Simulationszeit bei der HLA beantragen muss. Dieser wird ihr von der HLA gewährt, wenn diese sicherstellen kann, dass keine der von der Komponente abonnierten Variablen mehr einen Update mit zuweisungsseitigem Simulationszeitstempel vor der beantragten Simulationszeit erhalten kann. Über diesen — leistungsfähigen, aber augenscheinlich selten genutzten — Service kann temporale und Zustandskonsistenz zwischen Simulatoren mit gänzlich verschiedenen Simulationszeitmodellen erzielt werden, bspw. zwischen ereignisgetriebenen und zeitgesteuerten Simulationen. Aus diesem Grunde machen wir in unserer Kosimulation extensiv Gebrauch von den Zeitsynchronisationsservices der HLA, und zwar mit sehr feingranularer zeitlicher Auflösung: Die Simulationszeiten der Teilsimulatoren, von denen einer ein Echtzeitsimulator ist während die anderen mit einer virtuellen Simulationszeit arbeiten, werden bis auf $\frac{1}{50}$s synchronisiert. Dies ist hinreichend, weil der minimale zeitliche Abstand zweier aufeinanderfolgender Zustandsupdates aus derselben Komponente $\frac{1}{50}$s Simulationszeit beträgt; das Gleiche gilt für die minimale Samplingperiode für Komponenteninputs.

Der HLA-Standard 1516-2000 bietet mannigfaltige weitere, über die Kopplung von Simulatoren hinausgehende, Möglichkeiten. Prinzipiell können beliebige Softwareprodukte oder Hardwarekomponenten an einer Federation teilnehmen, da für einen Federate lediglich die Verfügbarkeit und semantikkonforme Bedienung einer HLA-Schnittstelle Bedingung für die Teilnahme an einer HLA-synchronisierten Simulation ist. Ein weiterer Vorteil der HLA ist wegen der standardisierten Schnittstelle die Flexibilität hinsichtlich der Austauschbarkeit seiner Federates. So könnte (und wird in IMoST-II) zum Beispiel der Fahrsimulator ST-Software jederzeit einfach gegen einen anderen Fahrsimulator ausgetauscht werden, ohne dass Veränderungen an den übrigen Komponenten der Kosimulation nötig würden. Voraussetzung dafür ist lediglich, dass der alternative Fahrsimulator auch die von den übrigen Simulationsteilnehmern abonnierten Daten zur Verfügung stellt. Die Vermittlung der entsprechenden Abonnements wird dabei dadurch unterstützt, dass die innerhalb einer Federation zwischen den Federates ausgetauschten Daten zuvor in einem Object Model Template (OMT) spezifiziert werden, wobei die Struktur dieses Templates ebenfalls im IEEE-Standard 1516-2000 festgelegt ist. Damit entfällt die bei nicht-dedizierten Kommunikationsme-

chanismen (bspw. Sockets) oftmals notwendige explizite Umwandlung zwischen unterschiedlicher Kodierungen von Datenpaketen.

Dieser praktische Vorzug war für die Auswahl von HLA als Kosimulationsbasis von IMoST allerdings zweitrangig; wesentlich war das konkurrenzlose Zeitmanagement für verteilte Simulationen. Bei den in vielen Projektkontexten stattdessen eingesetzten ad-hoc-Kopplungen diverser Simulatoren über Message-Passing muss die Funktionalität des Zeitmanagents entweder in Eigeninitiative nachimplementiert werden oder durch andere adäquate Mittel verhindert werden, dass ein Simulator Datenpakete doppelt sieht bzw. mehrere Datenpakete unterschiedlicher Simulationsteilnehmer in der unzutreffender Reichenfolge verarbeitet. Wird dies unterlassen, so entsteht keine global konsistente Simulationszeit und Kosimulationen werden aufgrund von Race-Conditions nichtdeterministisch, was ihren analytischen Wert schmälert.

4 Arbeitsbericht zur Realisierung der HLA-basierten Kosimulation

Die in IMoST realisierte Kosimulation ist sowohl in Hinblick auf die enge Kopplung der lokalen Simulationszeiten der einzelnen Teilsimulatoren (max $\frac{1}{50}$s relative Abweichung) als auch auf die unmittelbare Einbindung des interaktiven Simulink-Simulators (anstelle von compiliertem Code) außergewöhnlich. Letztere ist für ein Plattform zur modellbasierten Entwicklung von Assistenzsystemen einer Einbettung compilierten Codes gegenüber klar vorzuziehen.

Erste Vorarbeiten zur Realisierung des IMoST-Kosimulators fanden im Rahmen der Diplomarbeit von Herrn Stefan Puch statt [10]. Sie beschrieb die grundlegenden Funktionsprinzipien der HLA einschließlich der Servicedienste ihrer Runtime-Infrastructure (RTI). Die RTI ist die Softwareschicht, welche für die Ausführung einer HLA-Simulation notwendig ist. Die Arbeit realisierte eine prototypische HLA-Simulation nach dem HLA-Standard 1.3 mit einer RTI der Firma MÄK (Demolizenz) und zeigte, wie mittels der HLA-Toolbox der Firma ForwardSim [3] ein in MATLAB / Simulink entwickeltes Modell an eine RTI angeschlossen werden kann. Während in der prototypischen Implementierung der Diplomarbeit jedoch der bezüglich der dargestellten Fahrdynamik stark idealisierende Rennsimulator Torcs [1] verwendet wurde, wurde später im Projektkontext von IMoST zwecks besserer Vergleichbarkeit mit Fahrexperimenten beim Projektpartner DLR-Braunschweig der Fahrsimulator ST-Software eingebunden. Außerdem wurde im IMoST-Kontext basierend auf Performanzergebnissen verschiedener RTI-Implementierungen (MÄK, Gertico) und der Tatsache, dass die bislang einzig verfügbare freie RTI "Gertico" noch nicht den neueren Standard HLA 1516-2000 erfüllt, entschieden, eine kommerzielle RTI zu verwenden. Die Wahl fiel dabei auf eine RTI der Firma Pitch [9], die sich im Zusammenhang mit Support und zwei weiteren Tools, einem zur Erstellung von Object Model Templates (OMTs) und einem Werkzeug zum Mitprotokollieren von HLA-Simulationen, als im Preis-Leistungsverhältnis optimal herausstellte.

Die in IMoST angepeilte Gesamtsimulation sollte sich somit aus den in Kapitel 2 genannten Simulatoren einschließlich der RTI der Firma Pitch (pRTI-1516) nach dem HLA 1516-2000 Standard zusammensetzen. Das zuvor genannte Tool zum Aufzeichnen von HLA-Simulationen, der HLA-Rekorder, konnte im Entwicklungskontext als Federate zur Fehlersuche und -analyse benutzt werden und offerierte die Möglichkeit, die während einer Simulation aufgezeichneten Daten für spätere Analysen zur Verfügung zu stellen. Um die spezifizierte Simulationsplattform zu erhalten (vgl. auch Abbildung 3.), wurden folgende Arbeiten durchgeführt:

- Das kognitive Modell wurde mit einer HLA-Schnittstelle nach aktuellem HLA-Standard 1516-2000 versehen,
- für MATLAB / Simulink wurde eine Toolbox nach dem HLA Standard 1516-2000 angeschafft und in Interaktion mit dem Hersteller in ihrer Funktionalität optimiert,
- für den Fahrsimulator ST-Software wurde erstmals eine HLA-Schnittstelle implementiert.

Die vollständige Umsetzung der genannten Arbeitspunkte wurde dabei durch zahlreiche unerwartete Probleme, die im Folgenden einschließlich ihrer Lösungen näher betrachtet werden, immer wieder verzögert.

4.1 ST-Software

Da die ST-Software nicht als Quellcode vorliegt, besteht als einzige Möglichkeit zur Interaktion mit anderen Programmen die Verwendung von einfachen Socketverbindungen, welche von dieser Software unterstützt werden. In einer der ST-Software eigenen Scriptsprache können hierzu via Socketverbindung Simulationsdaten ausgelesen bzw. in die Simulation eingespeist werden. Zwar gibt es auch noch ein API für C++, auf dessen Basis Erweiterungsmodule für die ST-Software programmiert werden könen, diese Module ermöglichen jedoch keinerlei bidirektionale Kommunikation, da derartige Module nur entweder als reines „Output"-Modul (z.B. für Bildkanäle) oder als reines „Input"-Modul (z.B. für Pedalsätze) fungieren können. Für die Kopplung der ST-Software an die eingesetzte RTI wurde deshalb ein Wrapper implementiert, der die notwendigen Input- und Output-Socketverbindungen kapselt und gleichzeitig die Funktionalität einer HLA-Schnittstelle bereitstellt. Der ursprünglich intendierte Anwendungsbereich und die mangelnde Anpaasbarkeit der Closed-Source-Software an alternative Verwendungen bereitet ein weiteres Problem in Hinblick auf die zeitgetreue Kosimulation und damit auf die Reproduzierbarkeit von Simulationsläufen: Da der Simulator interaktiv in Realzeit, also mit näherungsweiser Übereinstimmung von Simulationszeit und Wall-Clock-Time läuft und die Scriptsprache keine Möglichkeit vorsieht, die Simulationsgeschwindigkeit zu steuern, kann die ST-Software bei der Verwendung des Zeitmanagements einer HLA-Simulation nur als *Time-Regulating*- und nicht auch als *Time-Constrained* Federate fungieren. *Time-Regulating* bedeutet hierbei, dass ein Federate den Zeitfortschritt anderer Federates beeinflussen kann und *Time-Constrained* beschreibt, dass der eigene Zeitfortschritt von anderen Federates beeiflusst werden kann. Folglich kann die ST-Software zwar den Zeitfortschritt anderer Federartes beeinflussen, ist selbst aber bezüglich ihres Zeitfortschritts nicht beeinflussbar. Ungeachtet dessen muss auch ein *Time-Regulating*-Federate seinen gewünschten Simulationszeitfortschritt zuvor bei der RTI anfragen und genehmigt bekommen, eine Tatsache, die ein weiteres Problem aufwirft. Die ST-Software kann zwar per Socketverbindung über die interne Scriptsprache die Zeit zur Verfügung stellen, die seit dem Simulationsstart vergangen ist, damit besteht aber lediglich eine konservative Abschätzung welche dem tatsächlichen Simulationslauf aufgrund der Latenzen im Kommunikationssystem mehr oder weniger stark hinterherhinkt. Erschwerend kommt hinzu, dass diese Referenzzeitstempel keine äquidistante Schrittgröße haben, sondern je nach Komplexität des zu simulierenden Szenarios Schwankungen von bis zu dem Doppelten der regulären Schrittgröße eines Samples aufweisen können.

Der implementierte HLA-Wrapper sieht für die genannten Probleme folgende Lösungen vor: Der sogenannte *Lookahead* (siehe [10]), der für einen *Time-Regulating*-Federate vorgeschrieben ist, wurde für verschiedene Szenarien experimentell bestimmt. Dabei wurde eine Unterapproximation aller beobachteten Schrittgrößen gewählt und als zusätzliche Absicherung wird bei jedem neuen Simulationsschritt überprüft, dass der *Lookahead* nicht verletzt wird.[2] Um Jitter aufgrund der variierenden Schrittgröße entgegenzuwirken, wurde eine sichere, aber möglichst wenig konservative Überapproximation der maximal erwarteten Schrittgröße als logischer Zeitschritt beim Beantragen des Simulationszeitfortschritts bei der RTI gewählt. Die Zeitstempel aus dem o.g. Script stellen also die tatsächliche Rechenzeit dar, auf die jeweils für den nächsten Simulationschritt dieser sicher überapproximierte Wert hinzuaddiert und als Zeitfortschritt beantragt wird. Tatsächlich kann der Federate dann aber seine Simulationszeit um einen kleineren Schritt voranschieben, wenn er die im zugestandene Rechenzeit nicht voll ausschöpft. Dies führt zu keinen Inkonsistenzen im HLA-Zeitmodell. Auf diese Weise werden die Schwankungen der tatsächlichen pro Frame der Simulation benötigten CPU-Zeit nach außen auf die logische Zeit des Federates übertragen. Da der Federate *time-regulating* ist, synchronisiert dies auch die logische Zeit der gesamten Föderation. Besonders zeitkritisch ist damit der erste Simulationsschritt nach der initialen Synchronisierung der Föderation. Da der erste Datensatz inklusive Zeitstempel aus dem Script der ST-Software zur Synchronisierung genutzt werden muss, kann das erste Zeitfortschrittsgesuch bei der RTI erst danach erfolgen, d.h. wenn der erste Schritt schon berechnet ist. Da für alle Zeitfortschrittsanfragen jedoch eine möglichst genaue Überapproximation der normalerweise zu erwartenden Schrittgröße benutzt

[2] Im Fehlerfalle werden entsprechende Meldungen im Konsolfenster generiert.

wird, bleibt für den ersten Schritt aller anderen Simulationsteilnehmer nicht so viel CPU-Zeit übrig, wie bei allen weiteren Simulationsschritten. Abbildung 2 veranschaulicht die Problematik etwas genauer: Die Daten der ST-Software gelangen via ST-Socket zum ST-HLA-Wrapper. Der

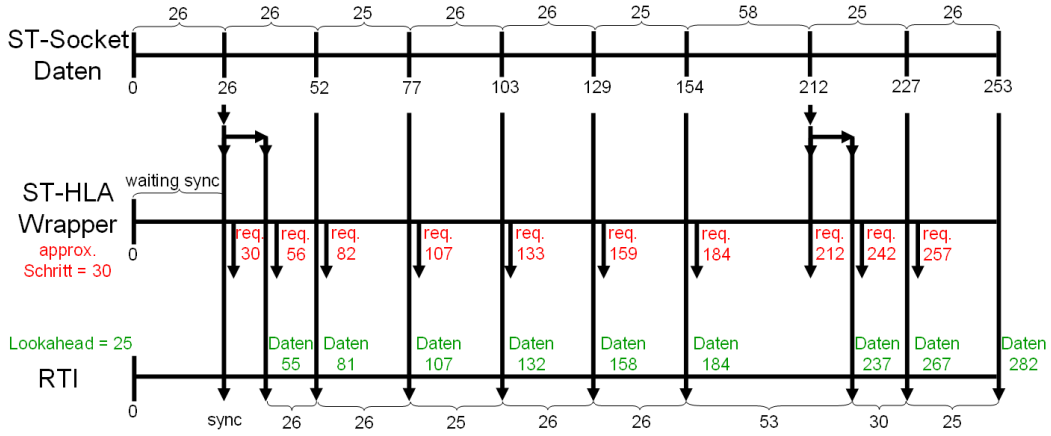

Abbildung 2. HLA-Schnittstelle der ST-Software

erste Datensatz wird einerseits benutzt, um die gesammte Föderation zu synchronisieren und andererseits, um den ersten Zeitfortschritt bei der RTI zu beantragen, was als *request* bezeichnet wird, in der Abbildung abgekürzt mit *req*. Für den ersten Simulationsschritt wird der überapproximierte Wert von beispielsweise 30 ms beantragt, in jedem weiteren Simulationsschritt wird dieser Wert zu dem Zeitstempel aus dem Script hinzuaddiert, so dass alle weiteren Simulationsschritte im Vorfeld beantragt werden. Der *Lookahead* wurde bei einer Schrittgröße, die im Mittel zwischen 25 ms und 27 ms liegt, mit 25 ms unterapproximiert. Der *Lookahead* wird beim Senden der Daten an die RTI zur jeweils aktuellen logischen Zeit des Federate hinzuaddiert. Die aktuelle logische Zeit der ST-Software entspricht in der Abbildung der jeweils beantragten Zeit. Auf die Darstellung der beim Beatragen eines Zeitfortschritts abzuwartenden *grant-Callback* wurde der Übersichtlichkeit halber verzichtet. Da der Federate jedoch nur *Time-Regulating* und nicht *Time-Constrained* ist, kann man sich diesen Callback als direkte Antwort auf die Anfrage vorstellen. Sollte auf Grund von Schwankungen der ST-Software der überapproximierte Wert von 30 ms doch einmal (wie in der Abbildung mit dort 58 ms) verletzt werden, so wird vor dem Senden dieses Datensatzes der Zeitfortschritt bei der RTI genau bis auf diesen Zeitstempel nachgeholt. Auf diese Weise wird verhindert, dass die an die anderen Simulationsteilnehmer zu sendenden Daten nur bis zu einem logischen Zeitpunkt gültig sind, welcher bereits in der Vergangenheit liegt.

4.2 HLA-Toolbox

Die HLA-Toolbox ist ein kommerzielles Erweiterungspaket für MATLAB von der Firma Forward-Sim [3], welches als closed Source zur Verfügung gestellt wird. Da es nach dem HLA-Standard 1516-2000 keine RTI Schnittstelle für MATLAB gibt, sondern nur für Hochsprachen wie Java oder C++, ist es die Aufgabe der HLA-Toolbox, diese Hochsprache zu kapseln und in einer von MATLAB aus benutzbaren Sprache (M-Code) zur Verfügung zu stellen. Genau genommen fungiert die HLA-Toolbox damit als Wrapper, indem sie die für einen Federate zur Kommunikation mit der RTI notwendigen Botschafter-Objekte zwischen C++ und M-Code übersetzt. So werden dem Benutzer in MATLAB Funktionen zur Verfügung gestellt, um Daten an die RTI zu senden und in umgekehrter Richtung gibt es ein Verzeichnis mit den diversen Callback-Funktionen, die von der RTI aufgerufen werden, um mit einem Federate zu kommunizieren. Aus diesen ebenfalls

in M-Code vorliegenden Funktionen besteht dann die Möglichkeit, die Daten von der RTI in einer eigenen MATLAB-Applikation nutzbar zu machen. Bedingt durch die aus der Diplomarbeit vorliegende HLA-Toolbox Version 1.3 (vergleiche Kapitel 4), wurde zunächst versucht mittels des zur RTI von Pitch gehörenden Pitch1516Adapter die Dusty-Deck-Toolbox an die pRTI-1516 zu koppeln. Der Adapter sollte nach Angabe des Herstellers eine Möglichkeit schaffen, Federates, die nach dem alten HLA 1.3 Standard entwickelt wurden, an eine RTI nach dem neueren Standard zu koppeln. Da jedoch die HLA-Toolbox zunächst einmal keine Möglichkeit vorsah, den genannten Adapter als „Version“ für eine RTI zu akzeptieren und auch die zu erwartenden Probleme, die Daten eines in MATLAB entwickelten Federate über zwei verschiedene Wrapper (HLA-Toolbox 1.3, Pitch1516Adapter) an eine RTI zu schicken, nicht minder werden würden, folgte die Entscheidung auch noch eine HLA-Toolbox für den HLA-Standard 1516-2000 zu beschaffen. Da es sich bei der HLA-Toolbox nach dem neueren HLA-Standard jedoch um eine Neuentwicklung des Herstellers handelte und es verschiedene RTIen mit unterschiedlichen Implementierungen gibt, mit denen die Toolbox funktionieren sollte, waren auf Grund unzureichender Softwaretests noch diverse Fehler (Bugs) in der HLA-Toolbox vorhanden, die analysiert und an den Hersteller berichtet werden mussten. So verging angefangen bei der Version 1.70 der HLA-Toolbox im Dezember des Jahres 2007 bis zur Version 1.92 Ende November 2008, die zunächst alle berichteten Probleme im Zusammenspiel mit der pRTI-1516 behob, fast eine ganzes Jahr. So waren beispielsweise Zeitstempel bei der Konvertierung von der RTI nach MATLAB fehlerhaft, Parameter in Funktionen undokumentiert, sowie auch Seiteneffekte auf andere Softwarekomponenten des Arbeits-PCs zu beheben. Die HLA-Toolbox in der Version 1.92 ermöglichte es dann im Rahmen des zweiten Meilensteins von IMoST im April 2009 erstmals einen in MATLAB / Simulink implementierten Federate sowohl *Time-Regulating* als auch *Time-Constrained* mit einer Simulationsschrittweite vom 50 ms in einer HLA-Simulation mitlaufen zu lassen.

4.3 Kognitives Modell

Das kognitive Modell ist eine Eigenentwicklung des Oldenburger Forschungs- und Entwicklungsinstitut für Informatik-Werkzeuge und -Systeme (kurz OFFIS). Ursprünglich aus einer Dissertation [6] zum Nachbilden von Pilotenverhalten entstanden, ist das Modell inzwischen weiter entwickelt worden und kann zum derzeitigen Stand auch mit Regeln geladen werden, um beispielsweise das Verhalten eines Autofahrers in bestimmten Situationen wie dem IMoST Szenario nachzubilden (vgl. Kapitel 1). Ein großer Vorteil bei diesem Simulator ist die Tatsache, dass er mit Sourcecode zur Verfügung steht, so dass bei den Anpassungen der HLA-Schnittstelle für den HLA-Standard 1516-2000 direkt auf alle notwendigen Details des Modells zugegriffen werden konnte. Dennoch bringt die Anbindung des kognitiven Modells vielfältige Probleme mit sich. Basierte die erste Version des Modells noch auf der Programmiersprache Prolog, so ist sie in der aktuellen Version in C implementiert, nicht jedoch in C++, wie vom 1516-2000 unterstützt. Damit müssen unterschiedliche, teilweise inkompatible Compiler, Laufzeitsystem und Bibliotheken in Verbindung gebracht werden. So wird das kognitive Modell derzeit unter dem Betriebsystem Windows in C mit dem MinGW Compiler und der *GLib* übersetzt, während die notwendigen Bibliotheken zur Kommunikation mit der pRTI-1516 nur für Microsoft Visual Studio bzw. den GNU gcc Compiler verfügbar sind. Leider gibt es bei den genannten Compilern keine einheitliche Verwendung von Compilermakros oder Pointerarithmetik, was die Übersetzung des Soucrecodes mit beiden Compilern deutlich erschwert. Letzendlich wurde die HLA-Schnittstelle für das kognitive Modell daher als Bibliothek (lib/dll) unter Visual Studio implementiert, gegen die das kognitive Modell selbst dann bei seiner Übersetzung gebunden wird.

5 Zusammenfassung der HLA-Simulation und Ausblick

Nach Beseitigung aller in den vorherigen Kapiteln genannten Probleme wurde zum Meilenstein 2 des IMoST Projektes im April 2009 die in Abbildung 3 dargestellte, verteile HLA-Simulation mit vier Federates vorgestellt, die dem HLA 1516 Standard entspricht. Als Federates kamen die

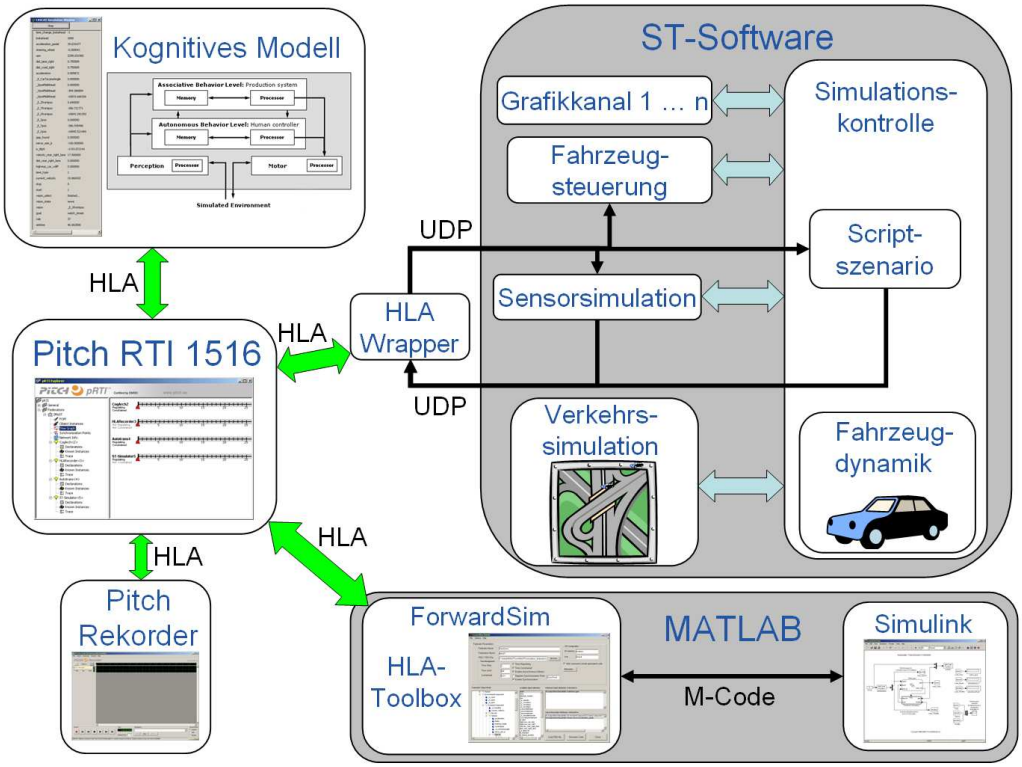

Abbildung 3. HLA-Simulation mit vier Federates

ST-Software, das kognitive Modell und die Entwicklungsumgebung MATLAB / Simulinkzum Einsatz. Als vierter Federate konnte der HLA-Rekorder die während der Föderation ausgetauschten Daten aufzeichnen. Für die Ausführung der Simulation wurden insgesamt sechs PCs und ein Laptop verwendet, die über ein 100 MBit LAN in einem eigenen Netz zusammengeschaltet wurden. Da die ST-Software für jedes Modul einen eigenen PC benötigt, entfielen vier der PCs auf die ST-Stoftware (Verkehrssimulation, Simulationskontrolle mit Scriptsteuerung, Fahrzeugsteuerung für den Input wie Gas, Bremse, Lenkwinkel, etc. und eine Sensorsimulation für den zusätzlichen Output von Simulationsdaten, die nicht über die Scriptsprache zugänglich waren). Jeweils ein weiterer PC wurde für die Pitch RTI sowie den HLA-Rekorder benötigt und ein Laptop für MATLAB / Simulink mit der HLA-Toolbox. Das kognitive Modell konnte auf dem gleichen PC wie die Sensorsimulation ausgeführt werden und der HLA-Wrapper auf dem gleichen PC mit der Fahrzeugsteuerung.

Auf der Basis dieser Simulationsplattform wurden seit dem Meilenstein 2 des Projektes Erweiterungen vorgenommen, die es in Zukunft erlauben sollen, die HLA-Simulation von außen steuerbar zu machen. Die zugrunde liegende Idee ist die Simulationszeit für ein vor der Simulation spezifiziertes Fahrerverhalten deutlich zu reduzieren. Möchte man beispielsweise das Verhalten des kognitiven Modells in bestimmten Situationen des IMoST-Szenarios (vgl. Kapitel 1) analysieren, so kann es bei einer normalen Simulation mitunter sehr lange dauern, bis die gewünschte Situation eintritt. Ein gezieltes Ansteuern der zuvor spezifizierten Situation würde eine Analyse deutlich vereinfachen. Damit für das Erkennen einer spezifizierten Situation kein Fachpersonal die HLA-Simulation beobachten muss, wurde auf der Basis einer mehrwertigen Temporalen Logik QLTL [11], [12] in ein HLAobserver [5] entwickelt, der während einer HLA-Simulation ein spezifiziertes Verhalten automatisch überprüfen kann. Mit dem HLAobserver erhält man nach jedem Simulationslauf eine Aussage, wie gut oder wie schlecht ein spezifiziertes Verhalten bzw. Ereignis erreicht wurde, was

sich beispielsweise für das Beobachten von kritischen Situationen während eines Simulationslaufes nutzen lässt. Zusatzlich wurde für das kognitive Modell eine Schnittstelle geschaffen, die es erlaubt, das bis dato teilweise zufällig generierte Verhalten eines Fahrers in einem weiteren Simulationslauf identisch nachzubilden, indem die benutzten Zufallsfunktionen bei Bedarf auch auf einen genau definierten Wert gesetzt werden können. Eine derzeit in Forschung befindliche Komponente, welche die Ergebnisse des HLAobservers automatisch auswertet, um in den folgenden Simulationsläufen sowohl das kognitive Modell als auch das simulierte Szenario mit entsprechenden Parametern zu versorgen, welche das spezifizierte Verhalten mit größere Wahrscheinlichkeit erreichbar machen, wird das Ziel einer heuristisch geführten HLA-Simulation deutlich näher bringen.

Literatur

1. Espie, Eric and Guionneau, Christophe. *TORCS Homepage*. URL: `http://torcs.sourceforge.net`. Stand: 20.03.2010.
2. Hardi Hungar et al. Imost homepage. Stand: 20.03.2010.
3. ForwardSim Inc., `http://www.forwardsim.com`. *ForwardSim Homepage*. Stand: 01.10.2009.
4. M. Fränzle, H. Hermanns, and T. Teige. Stochastic satisfiability modulo theory: A novel technique for the analysis of probabilistic hybrid systems. In Magnus Egerstedt and Bud Mishra, editors, *Proceedings of the 11th International Conference on Hybrid Systems: Computation and Control (HSCC'08)*, volume 4981 of *Lecture Notes in Computer Science*, pages 172–186. Springer, 2008.
5. Tayfun Gezgin. Observerbasierte on-the-fly Auswertung von QLTL-Formeln innerhalb eines HLA-Simulationsverbunds. Diplomarbeit, Carl von Ossietzky Universität Oldenburg, September 2009. under review.
6. Andreas Lüdtke. *Kognitive Analyse Formaler Sicherheitskritischer Steuerungssysteme auf der Basis eines integrierten Mensch-Maschine-Modells*, volume 288 of *Dissertationen zur Künstlichen Intelligenz*. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 1 edition, Februar 2005.
7. The MathWorks, `http://www.mathworks.de`. *MATLAB / Simulink / Stateflow*. Stand: 01.10.2009.
8. Roberto Passeron, Imene Ben Hafaiedh, Albert Benveniste, Daniela Cancila, Arnaud Cuccuru, Werner Damm, Alberto Ferrari, Sebastien Gerard, Susanne Graf, Bernhard Josko, Leonardo Mangeruca, Thomas Peikenkamp, Alberto Sangiovanni-Vincentelli, and Francois Terrier. Meta-models in Europe: Languages, tools and applications. *IEEE Design & Test of Computers*, 2009.
9. Pitch Technologies AB, `http://www.pitch.se`. *Pitch Homepage*. Stand: 25.03.2010.
10. Stefan Puch. Kosimulation von Verkehrssystemen und Operateuren. Diplomarbeit, Carl von Ossietzky Universität Oldenburg, Oktober 2007.
11. Gerald Sauter, Henning Dierks, Martin Fränzle, and Michael R. Hansen. Light-weight hybrid model checking facilitating online prediction of temporal properties. Technical report, Carl von Ossietzky Universität Oldenburg, 2009. `http://imost.informatik.uni-oldenburg.de`.
12. Gerald Sauter, Henning Dierks, Martin Fränzle, and Michael R. Hansen. Light-weight hybrid model checking facilitating online prediction of temporal properties. In *Proceedings of the 21st Nordic Workshop on Programming Theory, NWPT '09*, Kgs. Lyngby, Denmark, 2009. Danmarks Tekniske Universitet. Accepted.
13. IEEE Computer Society. *1516-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules*. Institute of Electrical and Electronics Engineers, Inc., Dezember 2000.
14. IEEE Computer Society. *1516.1-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification*. Institute of Electrical and Electronics Engineers, Inc., Dezember 2000.
15. IEEE Computer Society. *1516.2-2000 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification*. Institute of Electrical and Electronics Engineers, Inc., Dezember 2000.