SILAB und JDVE

Sven Müller
sven.mueller@informatik.uni-oldenburg.de

Seminar der Projektgruppe StreamCars
DLR
Universität Oldenburg
Department für Informatik
Abteilung Informationssysteme

Abstract:
In dieser Ausarbeitung werden SILAB und JDVE, zwei Fahrsimulatoren, beschrieben. Diese Beschreibungen werden zur vereinfachten Einordnung in einen Überblick über weitere Fahrsimulatoren eingebettet. Für jeden Fahrsimulator werden die Merkmale seiner technische Architektur ebenso betrachtet wie herausragende Eigenschaften.

1 Einleitung

Für die Simulation von Fahrzeugen im Straßenverkehr können grundsätzlich zwei verschiedenartige Ansätze ausgemacht werden, die sich durch ihren Betrachtungswinkel bzw. Schwerpunkt unterscheiden.

Ein Verkehrssimulator zielt darauf ab, vor Allem den Fahrzeugverkehr realitätsnah zu simulieren. Eine von dieser Art von Simulatoren erstellte Verkehrssimulation ermöglicht es, den Straßenverkehr zu analysieren und auf die Weise Erkenntnisse über die darin enthaltene Dynamik zu erlangen. Beispielsweise können Verkehrssimulatoren bei der Beantwortung der Frage helfen, unter welchen Umständen ein Verkehrsstau entsteht.

Liegt der Fokus des Simulators hingegen auf der realitätsgetreuen Simulation des einzelnen, eigenen Fahrzeugs, so handelt es sich um einen Fahrzeug- bzw. Fahrsimulator. Die von ihm verwirklichte Fahrzeug- bzw. Fahrsimulation kann z. B. dabei behilflich sein, ein Fahrassistenzsystem zu entwickeln.

Diese Ausarbeitung beschäftigt sich mit der näheren Beschreibung der Fahrsimulatoren SILAB und JDVE. Aus dem Bestreben heraus, eine vernünftige Einordnung dieser zu gewährleisten, wird jedoch in Abschnitt 2 zunächst eine größere Auswahl von Fahrsimulatoren, die SILAB und JDVE einschließt, kurz vorgestellt, bevor in den darauffolgenden Abschnitten 3 und 4 auf einige Aspekte von SILAB und JDVE genauer eingegangen wird, wobei einige Eigenschaften der jeweiligen Simulationsplattform (3.1 und 4.1) und die Benutzung (3.2 und 4.2) erklärt werden. Zum Schluss wird in Abschnitt 5 ein Fazit aus den vorangehenden Betrachtungen gezogen.

2 Fahrsimulatoren im Überblick

2.1 Kurzvorstellung von SILAB

SILAB ist eine Fahrsimulationsplattform, die vom Würzberger Institut für Verkehrswissenschaften (wivw)[1] entwickelt wurde. Laut wivw ist diese Plattform in einer kontinuierlichen Zusammenarbeit mit Kunden entstanden (u. a. Bosch und Continental).

SILAB zeichnet sich durch GUI-Orientierung und ausführliche Dokumentation aus, auch wenn letztere stellenweise unvollständig ist. Nichtdestotrotz sorgen diese Elemente für einen relativ einfachen Einstieg in die Verwendung der Simulationsplattform.

Zu den Eigenschaften von SILAB gehört weiterhin der umfassend modulare Aufbau. Die einzelnen Komponenten – von SILAB Digital Processing Units (DPUs) genannt – können beliebig verknüpft und ausgetauscht werden. Die DPUs können außerdem auf verschiedene Rechner verteilt werden, sodass eine einzige Simulation beliebig viele Rechner – und somit beliebig viel Rechenkapazität – involvieren kann. Welche DPUs in welcher Art und Weise in einer Simulation verwendet werden, wird in einfachen Textdateien festgelegt.

Ferner kann SILAB vielfältig erweitert werden. So ist es u. a. möglich, DPUs in Ruby und C++ zu entwickeln.

Eine nähere Betrachtung einiger Aspekte des JDVE findet in Abschnitt 3 statt.

2.2 Kurzvorstellung von SiVIC

SiVIC[2] ist ein Fahrzeug-, Infrastruktur und Sensorensimulator – gehört also zu den Fahrzeugsimulatoren (Vehicle (Simulation) Environment, VE) – und wurde von INRETS[3]-LIVIC[4] entwickelt [BIMIGI$^+$09].

Der Fokus von SiVIC liegt auf der prototypischen Entwicklung von virtuellen Sensoren, wobei eine möglichst realitätsgetreue und qualitativ hochwertige Simulation angestrebt wird. Es existiert eine große Auswahl an virtuellen Sensoren, auch von denen, die es zur Zeit noch nicht auf dem Markt gibt.

Allgemein ist es das Ziel, mit SiVIC das Fahrzeug und das Fahrzeugverhalten so exakt wie möglich abzubilden, um nicht auf ein reales Fahrzeug für die Sammlung von Daten und Evaluationsergebnissen angewiesen zu sein.

Von seiten der technischen Architektur arbeitet SiVIC mit Plug-ins, die die Simulation von Fahrzeugen, Sensoren und grafischen Objekten kapseln und dynamisch während des Simulationsablaufs eingebunden und entfernt werden können. U. a. zu diesem Zweck wurde die Verarbeitung einer eigens kreierten Skriptsprache implementiert, mit der zur Laufzeit

[1] `http://www.wivw.de/`, letzter Zugriff: 03.05.2010
[2] Simulateur Véhicule-Infrastructure-Capteurs, Vehicle-Infrastructure-Sensors Simulator
[3] Institut national de recherche sur les transports et leur sécurité
[4] Laboratory on the interactions between Vehicle-Infrastructure-Driver

die Simulation angepasst werden kann.

Weiterhin wurde in SiVIC eine Entkopplung der detailreichen grafischen Ausgabe von der Simulation selbst verwirklicht und es sind zwei Zeitmodi verfügbar: Zentralprozessortakt und virtuelle Zeit.

SiVIC bietet zusammenfassend eine hochqualitative Simulationsumgebung für Fahrzeuge, Sensoren und Umwelt. Um das eingebettete System des Fahrzeugs selbst jedoch zu simulieren, bedarf es der Anbindung an ein anderes System. Hierzu ist ^{RT}Maps [5] in der Lage: ^{RT}Maps ermöglicht die Entwicklung und Simulation eingebetteter Systeme und kann diese auf die technischen Elemente eines Fahrzeugs übertragen.

Durch die Verknüpfung von SiVIC und ^{RT}Maps ist es möglich, ein Fahrassistenzsystem zu entwickeln, indem die von SiVIC produzierten Sensorsimulationsdaten an das simulierte eingebettete System in ^{RT}Maps gereicht werden. Dieses analysiert die Daten, trifft ggf. die Entscheidung, zu handeln und schickt dem simulierten Fahrzeug in SiVIC entsprechende Befehle. Auf diese Weise kann ein Fahrassistenzsystem prototypisch entwickelt und virtuell getestet werden, um zu einem späteren Zeitpunkt durch Transfer des eingebetteten Systems auf das reale Fahrzeug zum Einsatz zu kommen.

2.3 Kurzvorstellung von COSMO-SiVIC

COSMO-SiVIC entstand durch Weiterentwicklung von SiVIC (vgl. 2.2), womit vordergründig die Integration des Fahrermodells COSMODRIVE gemeint ist. Erstellt wurde COSMODRIVE ebenso wie COSMO-SiVIC von INRETS[3]-LESCOT[6] im Rahmen des EU-Projekts ISi-PADAS[7] [BIMIGI$^+$09].

Die Erweiterung von SiVIC wird durch ein Plug-in realisiert. Es beinhaltet das Fahrermodell und weitere Anpassungen, die mit den Anforderungen des genannten Projekts ISi-PADAS zusammenhängen. So enthält COSMO-SiVIC eine neue grafische Schnittstelle und eine Datenbankanbindung zur Speicherung von Simulationsdaten. Mit COSMO-SiVIC ist es zudem möglich, mehrere Programminstanzen von SiVIC zu verwenden, wobei das Master-Slave-Prinzip Anwendung findet: eine Instanz ist der Master, der die Berechnungen übernimmt, während die Slaves lediglich die grafische Anzeige übernehmen. Die Synchronisation findet über UDP statt.

Des Weiteren bietet COSMO-SiVIC die Durchführung sekundärer Aufgaben für Testpersonen an. Hierzu können visuelle und auditative Ablenkungen verwendet werden und die so erhaltenen Ergebnisse in der Datenbank zur späteren Analyse abgespeichert werden.

COSMO-SiVIC soll zu einem späteren Zeitpunkt im Verlauf von ISi-PADAS mit JDVE (vgl. 2.5) verschaltet werden, um die jeweiligen Vorteile nutzen und die jeweiligen Nachteile ausgleichen zu können.

[5] Real Time, Multisensor, Advanced Prototyping Software

[6] Laboratory of Ergonomics and Cognitive Sciences Applied to Transport

[7] Integrated Human Modelling and Simulation to support Human Error Risk Analysis of Partially Autonomous Driver Assistance Systems

2.4 Kurzvorstellung von DOMINION

DOMINION wurde vom Deutschen Zentrum für Luft- und Raumfahrt (DLR) mit dem Ziel entwickelt, als Mittelschicht für die Entwicklung von Fahrassistenz- und -automationssystemen zu fungieren [SDNDHD+09]. Damit ist gemeint, dass Systeme, die auf Basis von DOMINION entwickelt werden, auf verschiedener Hardware ohne Änderungen in der Implementierung laufen können. Das macht es möglich, für ein neu zu entwickelndes Fahrassistenzsystem zunächst in einem realen Fahrzeug Daten zu sammeln, das System dann in Simulatoren und schließlich in realen Fahrzeugen zu testen.

DOMINION unterstützt die Entwicklung von Realzeitsystemen und bietet ferner Möglichkeiten an, um die Verwirklichung sicherer und verlässlicher eingebetteter Systeme zu gewährleisten.

Technisch wurde die Architektur von DOMINION durch das Konzept der Diensteorientierten Architektur (Service Oriented Architecture, SOA) inspiriert, das eine hohe Flexibilität und Modularität impliziert. Mit dieser Idee im Hintergrund entstanden diverse Module, von denen jedes eine bestimmte Aufgabe übernimmt bzw. einen bestimmten Dienst anbietet.

2.5 Kurzvorstellung des JDVE

Die Abkürzung JDVE steht für „Joint Driver-Vehicle Environment“. Dabei handelt es sich um eine Fahrsimulationsplattform, die (reale) Fahrer und Fahrzeuge ebenso unterstützt wie (virtuelle) Fahrer- und Fahrzeugmodelle.

Der Umstand, dass es eine „gemeinsame“ Plattform ist, soll verständlich machen, dass sie von allen Partnern des Ursprungsprojekts gemeinschaftlich verwendet werden soll. Das JDVE wurde im aktuell laufenden EU-Projekt ISi-PADAS[7] entwickelt.

In ISi-PADAS geht es darum, eine neuartige, innovative Methodik zur Unterstützung risikozentrierter Entwicklung und Prüfung teilweise autonomer Fahrassistenzsysteme hervorzubringen [new10]. Im Rahmen von ISi-PADAS wird das JDVE unter der Leitung des Deutschen Zentrums für Luft- und Raumfahrt (DLR) verwirklicht.

Technisch basiert das JDVE auf DOMINION (vgl. 2.4), nutzt allerdings nicht alle in DOMINION verfügbaren Module [SDNDHD+09].

Ebenso wie in DOMINION stellen die Module des JDVE allerdings allesamt eigenständige Programme dar und werden Applikationen genannt. Sie tauschen Daten über eine gemeinsame Zone namens Blackboard aus. Eins der Applikationen stellt den Server, auf dem sich die anderen Applikationen „einloggen“, und der die Simulation verwaltet.

Im Rahmen des Projekts wurde das JDVE erweitert. So wurde die Anforderung umgesetzt, eine Simulation mit einem variablen, synchronen Zeitablauf zu ermöglichen, und es wurden Ontologien integriert, die bspw. die zur Entwicklung von Assistenz- und Automationssystemen verfügbaren Sensordaten beschreiben [SDNDHD+09, BIMIGI+09]. Darüber hinaus befinden sich mit dem Stand von März 2010 die Integration von verschie-

denen Umwelt- und Fahrermodellen sowie teilweise autonomen Fahrassistenzsystemen in der Entwicklungs- bzw. Planungsphase [BIMIGI+09, new10].

Eine nähere Betrachtung einiger Aspekte des JDVE findet in Abschnitt 4 statt.

3 Nähere Betrachtung von SILAB

In diesem Abschnitt ist es das Ziel, einige Aspekte des zuvor kurz vorgestellten (vgl. 2.1) Fahrsimulators SILAB näher zu betrachten.

3.1 Eigenschaften

3.1.1 Projekte und Konfigurationsdateien

SILAB organisiert ausführbare Simulationen in Projekten, die in Unterordnern von `<Installationsverzeichnis>/Projects` abgelegt werden. Diese Projekte können in ein oder mehreren Konfigurationsdateien definiert werden, welche die Dateiendung `cfg` besitzen.

Um einen möglichst hohen Grad der Wiederverwendung von bereits erstellten Konfigurationen zu gewährleisten, bietet SILAB die Möglichkeit, innerhalb einer Konfigurationsdatei andere zu inkludieren. Hierzu wird an beliebiger Stelle in der Datei das Kommando `include` verwendet.

Mit Hilfe dieses Instruments kann eine Hierarchie von Konfigurationen erstellt werden, die eine Konfigurationsarchitektur entstehen lässt. Es ist im Normalfall allerdings nicht ratsam für eine neue Fahrsimulation eine neue Architektur aufzubauen. Stattdessen sollte die von SILAB bereitgestellte Standardkonfigurationsarchitektur verwendet werden. Weitere Informationen hierzu finden sich in [Wüi].

SILAB-Konfigurationsdateien können relativ unübersichtlich werden, wodurch sich leicht Fehler einschleichen können. Zum Zweck der Überprüfung einer Konfiguration auf Korrektheit dient das zum SILAB-System gehörende Hilfsprogramm „SILABCFGXCheck“. Weitere Informationen hierzu finden sich in [Wüf].

3.1.2 DPUs und der Aufbau von SILAB-Konfigurationsdateien

Bei den Konfigurationsdateien von SILAB handelt es sich um ASCII-Textdateien, die aus den Abschnitten `System`, `Configuration`, `TRF` und `SCN` bestehen können. `Configuration` teilt sich wiederum in die Unterabschnitte `Computerconfiguration <Rechnerkonfigurationsname>` und `DPUConfiguration <DPU-Konfigurationsname>` auf.

Der Konfigurationsabschnitt `System` beinhaltet die Konfiguration grundlegender Einstel-

lungen des SILAB-Systems, sie betreffen demnach das gesamte System bzw. die gesamte Simulation.

Der Abschnitt `Computerconfiguration` enthält die einzelnen Rechnerkonfigurationen. So wird festgelegt, welche Rechner bei der auszuführenden Fahrsimulation mitwirken sollen, wie sie heißen und wie sie erreichbar sind. Im Folgenden wird ein Beispiel zur Konfiguration eines Rechners gezeigt:

```
Computer LOCALHOST
{
  IP = "127.0.0.1";
  Frequency = 10;  # Takt (pro Sekunde)
  Operator = true; # Rechner ist "Operator" der Sim.
};
```

Die Frequenz gibt an, wie oft pro Sekunde jede vom Rechner verwaltete Softwarekomponente (DPU, s. u.) Ergebnisse berechnen soll. `LOCALHOST` ist der Name, über den auf den entsprechenden Rechner im Windows-Netzwerk zugegriffen werden kann. Ein Rechner ist „Operator“, also derjenige, der die Simulation bedient.

Wie der Name des Abschnitts `DPUConfiguration` bereits andeutet, geht es in diesem um die Konfiguration der an der Simulation beteiligten DPUs, den Digital Processing Units. Hierbei handelt es sich um die Softwaremodule, aus denen im SILAB-System eine Simulation erstellt werden kann. Dazu werden die gewünschten DPU-Typen bzw. -klassen instanziiert. Auf die Weise erhält man DPU-Instanzen.

DPU-Instanzen besitzen Ein- und Ausgänge, die mit den Ein- und Ausgängen anderer DPU-Instanzen verknüpft werden können. So entsteht ein Netzwerk aus DPUs, deren Zusammenspiel die Fahrsimulation entstehen lässt.

Um eine DPU zu instanziieren, ist der folgende Code im Konfigurationsabschnitt `Computerconfiguration` nötig:

```
DPUSCNX scn
{
   Computer = {LOCALHOST};
   Map = "Erster Versuch";
};
```

Es wird der DPU-Typ `DPUSCNX` instanziiert, wodurch die DPU-Instanz `scn` entsteht. Weiterhin wird für die DPU(-Instanz) angegeben, dass der Rechner `LOCALHOST` die Berechnungen übernehmen soll. Außerdem wird der Parameter `Map` gesetzt, wobei es sich um einen für diesen DPU-Typ spezifischen Parameter handelt. Solche und weitere DPU-Typ-spezifische Informationen können in den von SILAB mitgelieferten DPU-spezifischen Dokumentationen eingesehen werden.

Einen weiteren Teil des Konfigurationsabschnitts `DPUConfiguration` machen die Definitionen der DPU-Verknüpfungen aus. Die Syntax zum Verbinden eines Ein- und Ausgangs zweier DPUs verdeutlicht der folgende Code:

```
Connections =
{
   joy.SteeringWheel <-> vdyn.SteeringWheel,
   vdyn.Z <-> scn.Z
};
```

Hier werden zunächst zwei DPU-Instanzen namens `joy` und `vdyn` über zwei gleichnamige Ein- bzw. Ausgänge mit Namen `SteeringWheel` verbunden. Es folgt eine weitere Verknüpfung von `vdyn` und `scn` analog.

Die weiteren Konfigurationsabschnitte `TRF` und `SCN` beinhalten allgemeine Angaben zur Verkehrssimulation (`TRF`) und die Definition des Szenarios: Strecke, Umgebung, Verhalten der anderen Verkehrsteilnehmern usw. (`SCN`). Für eine Beschreibung dieser Elemente sei auf die SILAB-Dokumentation verwiesen.

3.1.3 SILABs Architektur

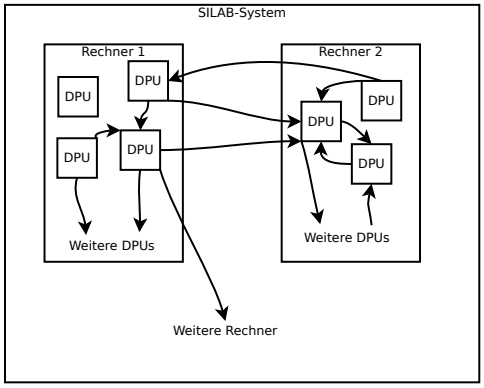


Abbildung 1: Schematische Darstellung der Architektur von SILAB

In SILAB wird durch die Konfigurationsdateien eines Projekts (vgl. 3.1.1) die Architektur einer Simulation definiert, indem festgelegt wird, welche DPUs für eine Simulation auf welche Art und Weise zusammenarbeiten sollen (vgl. 3.1.2), wobei jede DPU auf einem anderen Rechner ausgeführt werden kann. In Abbildung 1 wird dies veranschaulicht.

Es sei angemerkt, dass auch die Konfigurationsdateien durch Inkludieren weiterer Dateien eine Architektur beschreiben.

3.2 Benutzung von SILAB

3.2.1 Wie eine Simulation mit SILAB gestartet werden kann

Um eine Fahrsimulation zu starten wird das Hauptprogramm von SILAB verwendet. Mit diesem kann die gewünschte Fahrsimulation geöffnet werden, die dann von SILAB initialisiert bzw. geladen wird. Nach kurzer Zeit bietet das Programm die Möglichkeit, die Simulation zu starten, stoppen, pausieren, usw.

Um z. B. die mitgelieferte SILAB-Demo zu starten, kann mit dem Hauptprogramm die Datei `<SILAB-Installationsverzeichnis>/Projects/SILABDemo/SILAB_Demo_SGE.cfg` geöffnet werden[8]. Das eigene Fahrzeug ist nach dem Start der Simulation mit den Standardtastatureinstellungen steuerbar.

3.2.2 Wie eine Simulation mit SILAB erstellt oder anpasst werden kann

Um eine neue Simulation mittels SILAB zu erstellen, sollte zunächst ein neuer Projektordner im entsprechenden Oberordner von SILAB erstellt werden. In diesem können dann die Konfigurationsdateien der Simulation abgelegt werden. Durch Editierung dieser oder der anderer Projekte kann eine Simulation erstellt oder angepasst werden. Hier bieten sich viele Möglichkeiten. So können bspw. die Parameter der einzelnen DPUs verändert oder das Szenario angepasst werden.

3.2.3 Wie SILAB erweitert werden kann

Nutzung der vorhandenen DPUs zur Erweiterung von SILAB

Zum Lieferumfang von SILAB gehört eine große Anzahl von DPUs (Digital Processing Units, vgl. 3.1.2). Durch intelligente Zusammenschaltung von diesen lässt sich die Funktionalität einer Fahrsimulation unter SILAB bereits in Maßen an eigene Bedürfnisse anpassen, sodass die Erweiterung mittels beispielsweise Ruby überflüssig werden. Vor Allem die universellen DPUs bieten sich zu diesem Zweck an, da sie diejenigen sind, die Basisfunktionalitäten wie z. B. einfache logische und mathematische Operationen zur Verfügung stellen. In der Tat lassen sich mit diesen DPUs sogar einfache Fahrassistenzsysteme implementieren (vgl. [Wüd]).

Nutzung von Silascript und Ruby in SILAB

Falls die Erweiterung von SILAB durch Zusammenschaltung von DPUs (vgl. 3.2.3) für einen Anwendungsfall unangemessen bzw. gar nicht möglich sein sollte, die Nutzung von komplexeren Programmiersprachen wie Ruby oder C++ aber überzogen wäre, bietet sich Silascript an.

[8] Als Einstellungen können z. B. „Einzelplatz“ für „ComputerPool“ und „Full“ für „DPUPool“ gewählt werden.

Silascript ist eine einfache, prozedurale, Touring-vollständige Programmiersprache, die für die Programmierung einfacher DPUs gedacht ist [Wüh]. Zum Sprachumfang gehören allgemeine Konstrukte wie Variablen, Konstanten, Funktionen usw.

Im Folgenden wird ein Beispielskript für die Verdopplung eines Zahlenwertes gezeigt:

```
# Schnittstelle:
import var number input;  # Eingang ("Input" in SILAB)
export var number output; # Ausgang ("Output" in SILAB)

# Diese Funktion verdoppelt den Wert einer Zahl
number doppelt(number x) { return x * 2; };

# Ein- und Ausstiegsfunktion von SILAB
boolean Trigger(number a, number b)
{
  output = doppelt(input);
  return true; # false: Fehler, Programmstopp
};
```

Wird das obige Silascript in einer Textdatei abgelegt, kann es mit einer Instanz der DPU `DPUScript` angesteuert werden [Wüb], indem dieser als Parameter `Script` der Dateisystempfad zum Skript übergeben wird.

Weitere Informationen zur Nutzung von Silascript und im Rahmen von SILAB bieten [Wüh, Wüb].

Die Nutzung von Ruby-Skripten geschieht analog zu der von Silascript. Hierzu sei auf [Wüg, Wüa] verwiesen.

Nutzung von C++ in SILAB

Die mit Abstand meisten Möglichkeiten der Erweiterung bietet die Nutzung von C++. Ebenso wie in Silascript und Ruby (vgl. 3.2.3) kann beliebiger Code ausgeführt werden. Und ebenso wie in Ruby stehen dem Entwickler auf die Weise die Objektdatenbank und das Datenmodell von SILAB zur Verfügung. Allerdings kann mittels C++ zusätzlich die Menge der in SILAB verfügbaren DPU-Typen erweitert werden.

Um die Programmierung zu vereinfachen, bietet SILAB hierfür einen Assistenten an („SILABDPUWizard“), der ein Grundgerüst generiert [Wüe].

3.2.4 Wie externe Programme an SILAB angebunden werden können

In 3.2.3 wurden verschiedene Erweiterungsmöglichkeiten von SILAB gezeigt, die implizieren, dass u. a. mittels C++ und Ruby auf native Weise externe Programme angebunden werden können. Es können sowohl in C++ als auch in Ruby beispielsweise UDP-Verbindungen implementiert werden.

Allerdings vereinfacht SILAB durch mitgelieferte DPUs die Anbindung. Hier sei beispielhaft die DPU `DPUSocket` genannt, die Socketverbindungen aufbauen und nutzen kann [Wüc].

4 Nähere Betrachtung des JDVE

In diesem Abschnitt ist es das Ziel, einige Aspekte des zuvor kurz vorgestellten (vgl. 2.5) Fahrsimulators JDVE näher zu betrachten.

4.1 Eigenschaften des JDVE

4.1.1 Umweltmodelle im JDVE

Ein wichtiger Bestandteil des JDVE, der nicht aus der Entwicklung von DOMINION stammt, besteht aus verschiedenen Umweltmodellen. Diese teilen die im JDVE zu repräsentierende Umwelt nach Beweglichkeit und Veränderlichkeit der Objekte ein.

Die unbewegliche, unveränderliche Umwelt umfasst Elemente wie Häuser und Straßenbegrenzungen. Diese bleiben über die gesamte Simulation hinweg am selben Ort. Für sie wird keine logische Repräsentation benötigt, da sie für die Simulation – mit Ausnahme ihres Aussehens – irrelevant ist[9]. Deswegen existiert für diese Objekte lediglich das grafische Modell, das von der Applikation für die Umweltrepräsentation `EnvironmentManager` festgelegt und von Applikationen zur grafischen Ausgabe wie bspw. `NexGenViewer` gerendert wird.

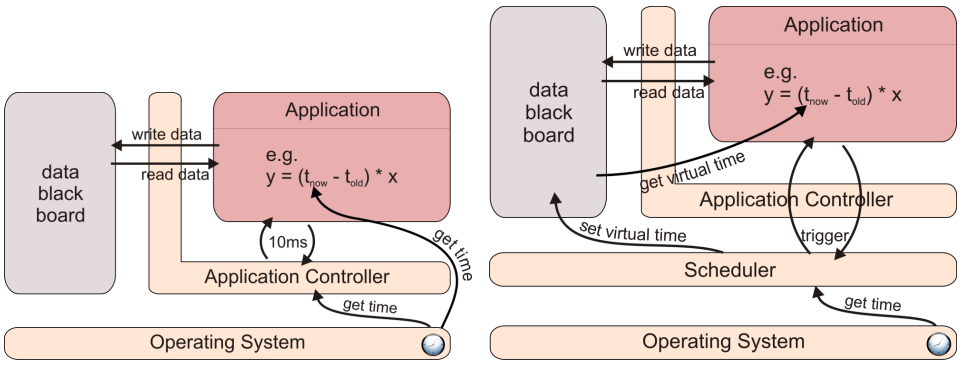
Die unbewegliche, dafür aber veränderliche Umwelt besteht u. a. aus Ampeln und Verkehrsleitsystemen, die sich zwar von ihrem Ausgangsort während der Simulation nicht fortbewegen, wohl aber ihren Zustand ändern. Diese Zustandsänderung ist für weitere Objekte der Simulation relevant (wie z. B. das Schalten der Ampeln für Verkehrsteilnehmer), weswegen zusätzlich zur grafischen eine logische Repräsentation erforderlich ist. Die Applikationen `EnvironmentManager` und `DynamicObjectSimulation` enthalten den logischen Status der unbeweglichen, veränderlichen Umwelt.

Zur beweglichen Umwelt gehören all jene Elemente der Umwelt, die dazu imstande sind, sich zu bewegen. In diesem Zusammenhang sind vor Allem das eigene Fahrzeug und der es umgebende Verkehr zu nennen. Für die logische Repräsentation der Objekte dieser Umwelt sind `DynamicObjectSimulation` und `DominionNLTwoTrack` verantwortlich, wobei das letztgenannte Modul das Modell des eigenen Fahrzeugs umsetzt, das eine höhere Komplexität aufweist als das der anderen Fahrzeuge.

[9] Eine Ausnahme stellt das Straßengeflecht dar.

4.1.2 Synchroner Zeitablauf im JDVE

Es besteht die Möglichkeit, Simulationen mittels des JDVE in einem synchronen Zeitablauf stattfinden zu lassen. Zum besseren Verständnis hiervon wird im Folgenden zunächst der Standardzeitablauf von DOMINION erklärt wie ihn Abbildung 2(a) verdeutlicht.



(a) Asynchroner Zeitablauf (b) Synchroner Zeitablauf

Abbildung 2: Realisierung des Zeitablaufs in JDVE . Aus [SDNDHD+09].

Standardmäßig wird die Funktionalität jede an einer Simulation beteiligte Moduls Applikation von der jeweiligen Modulverwaltungseinheit („Application Controller“) in einem Abstand ausgelöst, der für jedes Modul individuell einstellbar ist (in der Abbildung 10ms). Die Module laufen demnach konzeptuell asynchron.

Um in einer Simulation alle Module zu synchronisieren, wird eine zentrale Kontrollinstanz für den Zeitablauf, genannt Scheduler, eingesetzt (siehe Abbildung 2(b)). Er teilt den Zeitablauf in „Runden“ auf, in denen er die Funktionalität jeder Applikation genau einmal auslöst – in geordneter Reihenfolge. Für jede Runde legt er die virtuelle Zeit fest und macht sie im gemeinsamen Speicherbereich aller Applikationen für diese verfügbar.

Auf diese Weise ist es möglich, alle Applikationen gleich langsam oder schnell arbeiten zu lassen, wobei letzterer Fall von besonderem Interesse ist, da so langwierige Simulationen in kürzerer Zeit durchgeführt werden können.

Die Implementierung des synchronen Zeitablaufs umfasst zum einen den zusätzlichen DOMINION-Serverzustand `triggered` und zum anderen die Applikation `isiPADAS Scheduler2`.

4.1.3 Spezieller DOMINION-Datenkern des JDVE

DOMINION enthält einen Datenkern, der die verwendeten Variablen und Aufgaben in der In-Vehicle Service Description Language (VSDL) beschreibt. Im Zuge der Entwicklung des JDVE wurde dieser an die speziellen Anforderungen des Projekts ISi-PADAS angepasst [SDNDHD+09, BIMIGI+09]. So wurden z. B. Variablen eingeführt, die bestimmte

Fahrassistenzsysteme repräsentieren.

Da der Datenkern Teil DOMINIONs ist, wird in dieser Ausarbeitung nicht näher auf diesen Aspekt eingegangen.

4.1.4 JDVEs Architektur

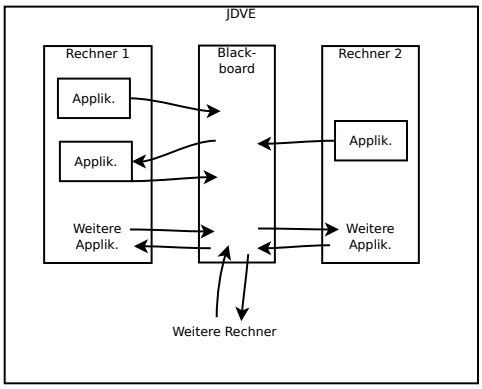


Abbildung 3: Schematische Darstellung der Architektur des JDVE

Die Architektur des JDVE, visualisiert in Abbildung 3, ist dieselbe wie die von DOMINION: Die Module, genannt Applikationen, kommunizieren nicht direkt, sondern indirekt über eine gemeinsame Speicherzone namens Blackboard[10]. Anders als bei SILAB sind die Module nicht Teil des Programms; die Module selbst sind Programme.

4.2 Benutzung des JDVE

4.2.1 Wie eine Simulation mit dem JDVE gestartet werden kann

Jede Applikation des JDVE besitzt eine eigene ausführbare Datei im Verzeichnis `<Installationsverzeichnis>/bin`. Da jede von ihnen ein eigenständiges Programm ist , kann sie unabhängig von anderen Applikation gestartet und gestoppt werden. Es bietet sich die Nutzung einer Stapelverarbeitungsdatei bzw. eines Startskripts an, die bzw. das die für einen bestimmten Anwendungsfall benötigten Applikationen in geordneter Reihenfolge startet. Um einzelne Applikationen verzögert starten zu lassen, wird das Programm `wait` mitgeliefert, das im selben Ordner zu finden ist wie die Startdateien der Applikationen.

[10] Eine Ausnahme stellt u. a. der DOMINION-Server dar, der mit den anderen Apllikationen auch auf direktem Weg Daten austauscht

4.2.2 Wie eine Simulation mit dem JDVE erstellt oder anpasst werden kann

Eine Simulation des JDVE lässt sich zunächst durch passende Wahl der benutzten Applikationen erstellen oder anpassen. Des Weiteren sind bestimmte Applikationen durch Konfigurationsdateien o. Ä. anpassbar. So erhält beispielsweise das Applikation `DynamicObjectSimulation` Angaben über das zu simulierende Verhalten in Form einer XML-Datei im Ordner `<Installationsverzeichnis>/Data` bzw. einem Unterordner davon.

4.2.3 Wie das JDVE erweitert werden kann

Zur Erweiterung des JDVE kann eine neue Applikation in der Programmiersprache C++ geschrieben werden, wofür der Dominion Base Application Assembler (DominionBAAL) genutzt werden kann. Er generiert automatisch das Grundgerüst für ein neues Modul, das direkt zur Programmierung der beabsichtigten Funktionalität verwendet werden kann.

An dieser Stelle sei auf Arbeiten über DOMINION selbst verwiesen, da sich das JDVE diesbezüglich auf die Funktionen und Programme von DOMINION stützt.

4.2.4 Wie externe Programme an das JDVE angebunden werden können

Wie in 4.2.3 erklärt, kann eine neue Applikation für das JDVE in C++ geschrieben werden. Insofern können alle Bibliotheken verwendet werden, die mit C++ angesprochen werden können. Um externe Programme an das JDVE anzubinden, kann also z. B. auf UDP zurückgegriffen werden, da C++-Bibliotheken hierfür existieren.

Tatsächlich liefert das JDVE bereits UDP-Funktionalität auf abstrakter Ebene mit. So ist eine Funktion innerhalb der von DominionBAAL generierten Klasse verfügbar, die UDP-Nachrichten nach belieben versendet und eine weitere, die UDP-Nachrichten entgegennimmt. Beide entstammen dem Softwareumfang von DOMINION.

5 Fazit

Es wurden in Abschnitt 2 einige Fahrsimulationsplattformen kurz vorgestellt. Diese Vorstellungen deuteten die Vielfalt in diesem Bereich an, sie unterscheiden sich je nach Anforderungen und nach Entwicklungsstand im Detail stark.

Auf die einzelnen Vorstellungen folgend wurden in den Abschnitten 3 und 4 SILAB und JDVE näher betrachtet.

SILAB bietet durch seine GUI-orientierten Programme einen schnellen und einfachen Einstieg. Auch die Anpassung oder Erstellung einer Simulation gestaltet sich in SILAB relativ problemlos. Von den Modulen, genannt DPUs, existiert eine große Anzahl für vielfältige Anwendungsfälle. Sie sind – wenn auch stellenweise unvollständig – ausführlich dokumentiert, was ebenso auf SILAB selbst und die mitgelieferten Werkzeugprogramme zu-

trifft. Eine eigene DPU für SILAB zu erstellen, ist knifflig, aber hierzu wird durch einen Assistenten Hilfestellung angeboten, der ein Grundgerüst für die Programmierung in C++ erstellt. In vielen Fällen ist ein solcher Aufwand nicht nötig, da bereits viele vorgefertigte DPUs zum Lieferumfang gehören.

Beim JDVE handelt es sich um ein Produkt mit starkem Entwicklungscharakter. Aus diesem Grund fällt der Zugang zum JDVE naturgemäß nicht leicht. Um eine Simulation zu starten, müssen die einzelnen Module bzw. Applikationen gestartet werden, die meist nicht über eine grafische Oberfläche verfügen.

Um eine Simulation im Rahmen des JDVE anzupassen, können die Applikationen entsprechend gewählt oder parametrisiert werden. Auch kann eine Applikation neu erstellt werden, wobei auf ein Werkzeug zur automatischen Generierung zurückgegriffen werden kann. Dieses ist zu großen Teilen selbsterklärend und bietet große Freiheiten in der Gestaltung, was vor Allem durch den speziellen modularen Charakter des JDVE begründet ist (jedes Modul, genannt Applikation, stellt ein eigenständiges Programm dar), sie bietet ein hohes Maß an Flexibilität, da zur Laufzeit Applikationen der Simulation hinzugefügt und entfernt werden können.

Ebenfalls herausgestellt werden sollte beim JDVE der Vorteil, das die auf der Basis dieses Simulators programmierten Funktionalitäten auf verschiedensten Betriebssystem- und Hardwareplattformen funktionieren. Die Nutzung eines Fahrassistenzsystems wird so beispielsweise ebenso auf einem physischen Simulator wie auch in einem realen Fahrzeug möglich. Dies ist darauf zurückzuführen, dass das JDVE auf DOMINION basiert, welches exakt mit Blick auf diese Möglichkeiten entwickelt wurde.

Abschließend lässt sich sagen, dass der Einstieg in SILAB leichter fällt als in JDVE. Die Anpassung und Erstellung einer Simulation gestaltet sich ebenfalls einfacher. Allerdings zeichnet sich das JDVE durch hohe Hardware-Abstraktion und den speziellen Modularitätscharakter aus.

Literatur

[BIMIGI+09] T. Bellet (INRETS), P. Mayenobe (INRETS), D. Gruyer (INRETS), J. C. Bornard (INRETS), J. Schindler (DLR), T. Krehle (DLR), Jens Alsen (OFFIS) und Andreas Lüdke (OFFIS). Joint-DVE Simulation Platform for phase 1. ISi-PADAS-Bericht, "Deliverable"4.2, Dezember 2009. Vertraulich.

[new10] ISi-PADAS-Newsletter 1. `http://www.isi-padas.eu/newsletter/ISi-PADAS_Newsletter1.pdf`, März 2010. Letzter Zugriff im April 2010.

[SDNDHD+09] Julian Schindler (DLR), Ulf Noyer (DLR), Christian Harms (DLR), Frank Flemisch (DLR), Aladino Amantini (Kite Solutions), Dominique Gruyer (INRETS-LCPC), Malte Zilinski (OFFIS), Fabio Tango (CRF) und Frank Köster (DLR). Ontology and Basic Version of the Joint DVE Simulation Platform. ISi-PADAS-Bericht, "Deliverable"4.1, Mai 2009. Vertraulich.

[Wüa] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: DPU-RubyScript*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüb] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: DPUS-cript*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüc] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: DPU-Socket*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüd] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Erste Schritte*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüe] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Erweiterung um DPUs*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüf] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Konfiguration und Bedienung*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüg] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Ruby API*. SILAB-Version 2.5, Stand: Oktober 2007.

[Wüh] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Silas-cript*. SILAB-Version 2.5, Stand: Januar 2009.

[Wüi] Würzburger Institut für Verkehrswissenschaften (wivw), Ansprechpartner: Prof. Dr. Hand-Peter Krüger, Raiffeisenstraße 17, 97209 Veitshöchheim. *SILAB: Standard-Konfigurationsarchitektur*. SILAB-Version 2.5, Stand: Januar 2009.