Die High Level Architecture: Anforderungen an interoperable und wiederverwendbare Simulationen am Beispiel von Verkehrs- und Infrastruktursimulationen

Ulrich Klein / Steffen Straßburger
Universität Magdeburg, Institut für Simulation und Graphik
Postfach 41 20
D-39016 Magdeburg
Email: {uklein@isg, strassbu@sunpool}.cs.uni-magdeburg.de

Der Beitrag stellt die High Level Architecture vor, die die Erstellung interoperabler und wiederverwendbarer Simulationsmodelle unterstützt. An einem Verkehrsbeispiel wird eine zivile Nutzung der aus dem militärischen Bereich stammenden HLA beschrieben und die Anforderungen aufgezeigt, die bei der Nutzung von HLA an klassische Simulationswerkzeuge gestellt werden.

1 Einführung

Mit dem Erscheinen der neuen High Level Architecture (HLA) des amerikanischen Defense Modeling & Simulation Office (DMSO) erhält das Gebiet der verteilten Simulation neue Impulse [1]. Die Arbeit beschäftigt sich mit der flexiblen verteilten Modellierung von Simulationen und allgemeineren Systemen sowie möglichen neuen Anwendungsfeldern, die durch HLA für Simulationen erschlossen werden können. Anhand des Beispiels einer Verkehrssimulation wird dargelegt, wie eine nach beteiligten Teilsystemen modularisierte Simulation aus einem herkömmlichen monolithischen Modell erstellt werden kann. Dabei werden die Anforderungen, die HLA an konventionelle Simulationstools stellt, beschrieben und derzeitige Grenzen aufgezeigt.

2 Die High Level Architecture (HLA)

Die HLA bietet einen Ansatz, die Forderungen nach Wiederverwendbarkeit und Interoperabilität von Simulationsmodellen umzusetzen und die Schwierigkeiten mit monolithischen Modellen bei Änderungen der Funktionalität oder Konnektivität zu umgehen.

Ähnlich dem CORBA zugrundeliegenden Prinzip wird eine einheitliche Schnittstelle (*HLA Interface*) definiert, welche die Teilnehmer eines gemeinsamen Simulationslaufs (*Federates*) aufzuweisen haben, um in Kontakt mit der *Runtime Infrastructure* (RTI) zu treten, welche Basis-, Koordinations- und Kommunikationsdienste zur Laufzeit bereitstellt. Eine *Federation* kann als ein Vertrag zur Durchführung eines Simulationslaufes (*Federation Execution*) zwischen den Federates angesehen werden, in dem die Einzelheiten und Objektmodelle der Federates (*Simulation Object Model*) und der Federation (*Federation Object Model*) festgelegt sind. Diese Informationen sind in einer definierten

Form zu dokumentieren (*Object Model Template*). Weiterhin legt HLA zwingende Verhaltensregeln für Federates und Federations fest. Ein Vorteil der Architektur ist die Offenheit für andere Arten von Federates; sowohl Softwarebausteine wie z.B. Datenbanken und Beobachter als auch reale Elemente wie Sensoren und andere Hardware können bei Einhaltung der HLA-Spielregeln bei Federation Executions mitwirken (Bild 1).

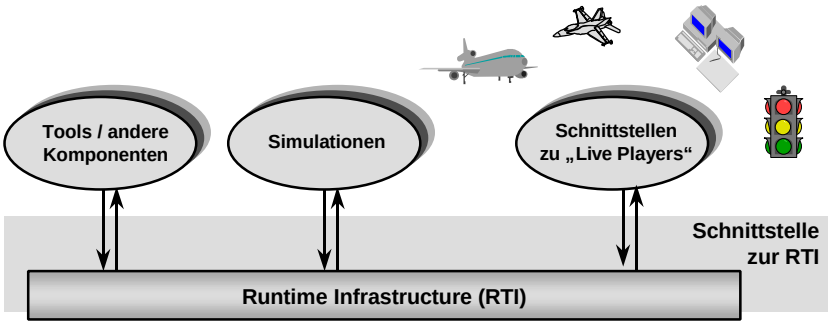

Bild 1 Aufbau der High Level Architecture

Ebenso wird ein transparentes Zeitmanagement unterstützt, welches den Federates außer der Koordinierbarkeit keine Vorschriften bzgl. des lokalen Zeitfortschritts (z.B. Zeitschritt-, Ereignis-, Echtzeitsimulation) macht.

3 Anwendungsbeispiel Verkehrs- und Infrastrukturmanagement

Unter Anwendung der HLA ist es möglich, ein System z.B. in Analogie zu dessen Subsystemen zu modellieren. Ein Beispiel für komplex vernetzte Subsysteme ist der Verkehr mit den zugrundeliegenden Infrastrukturen [4].

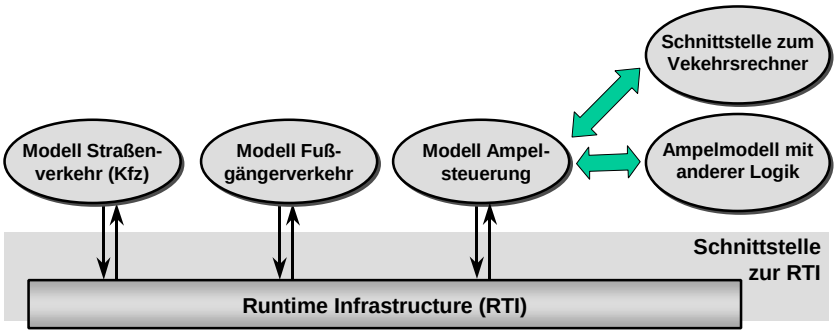

Bild 2 Aufbau des HLA-Beispiels einer Verkehrsregelung

So lassen sich bereits am Beispiel eines lichtsignalgesteuerten Verkehrsknotens Vorzüge der HLA zeigen (Bild 2): hierzu wurden jeweils getrennte Module geschaffen, die separat

den komplexen Verkehrsfluß von Autos, Radfahrern und Fußgängern modellieren. Eine weitere Komponente (Federate) enthält das Modell der Lichtsignalanlage (LSA) mit einer gewissen Steuerungslogik.

Durch die Flexibilität der High Level Architecture ist es möglich, dieses Ampel-Federate (aus Sicht der anderen Federates eine Black Box mit einheitlicher Schnittstelle und bekanntem Objektmodell) durch ein beliebiges anderes, z.B. mit einer anderen Steuerungslogik oder mit einer Schnittstelle zu einem Verkehrsrechner, auszutauschen.

4 Anforderungen der HLA an beteiligte Simulationen

Die oben erwähnten Vorzüge und Vorteile der HLA stellen teilweise beachtliche Anforderungen an die potentiellen Federates, die nachfolgend aus konzeptioneller und programmierungstechnischer Sicht beschrieben werden.

4.1 Konzeptionelle Anforderungen

In der verteilten Simulation können voneinander abhängige Simulationen ihren lokalen Zeitfortschritt nicht mehr autonom durchführen; vielmehr ist dieser (bei HLA mittels der RTI) zu koordinieren [2, 3]. So ist es z.B. bei ereignisgesteuerten Simulatoren nötig, die Ereignisketten von sich am Zeitfortschritt beteiligenden Federates miteinander zu synchronisieren bzw. verknüpfen. Es muß in einer gewissen Weise sogar die Möglichkeit bestehen, externe Ereignisse, die im lokalen Federate von Bedeutung sind, in die lokale Ereigniskette aufzunehmen. Dies ist z.B. nötig, wenn sich Attributswerte im externen Federate ändern, die im lokalen Federate von Interesse sind.

Eine weitere konzeptionelle Anforderung betrifft den Datenaustausch zwischen Federates sowie die Notwendigkeit der Darstellung externer Daten (Objekte, Attributswerte) im Federate (*ghosten*). Das HLA Konzept sieht vor, daß ein Federate für sich interessante Daten, die durch andere Federates modelliert werden, „abonniert“ und in der Folgezeit über Änderungen bezüglich dieser Daten informiert wird und diese ggf. intern abbildet.

4.2 Programmierungstechnische Anforderungen

Die Zusammenarbeit von Federates mit der RTI basiert auf einem Botschafterprinzip: dabei entsenden sowohl die RTI als auch das Federate ein Botschafter-Objekt zum jeweils anderen Kommunikationspartner; der RTI-Botschafter beim Federate nimmt Anfragen (Methodenaufrufe) entgegen, während die von der RTI gerufenen Dienste des Federate dem Federate-Ambassador beim RTI übergeben werden.

Die RTI-Software besteht (in der F.0-Version) aus dem RTIExec-Prozeß (der mehrere Federation Executions verwalten kann) sowie dem FedEx-Prozeß (der individuell für jede Federation Execution gestartet wird und dann die Kommunikation mit den Federates übernimmt). Außerdem muß ein potentielles Federate in der Lage sein, eine bereitgestellte RTI-Bibliothek einzubinden, welche das RTI-Ambassador-Objekt sowie das abstrakte Federate-Ambassador-Objekt enthält, wobei letzterer noch auszuimplementieren ist. Zur Zeit geschieht dies in C++; die Unterstützung von CORBA und weiteren Sprachen wie Ada95 und Java ist angekündigt.

Unsere Untersuchungen zeigten, daß es weniger die konzeptionellen, sondern vielmehr die programmierungstechnischen Forderungen sind, die potentielle Federates vor schwierige Aufgaben stellen.

5 Realisierung der Anbindung klassischer Simulatoren an HLA

Auf der Grundlage der oben genannten Anforderungen wurde stufenweise ein verteiltes Modell geschaffen. Auf dem Weg vom klassischen monolithischen Modell zu einem HLA-konformen verteilten Modell wurden in Zwischenstufen die Schnittstellen zwischen den Komponenten herausgearbeitet und zunächst mit einer nicht HLA-konformen Verteilung experimentiert. Das Modell beinhaltet die drei in Bild 2 dargestellten Basiskomponenten, die in der Endstufe als eigenständige Federates agieren. Für die Untersuchungen verwendeten wir die Simulationstools GPSS/H und SLX.

5.1 Zeitliche Synchronisation

In unserem Beispiel wurde sowohl eine zeitschritt- als auch eine ereignisgesteuerte Variante erstellt; eine Echtzeitkomponente ist vorgesehen. Unsere Lösung sieht das Einführen einer Planungsroutine bzw. eines Scheduling-Threads vor, der nebenläufig zur eigentlichen Simulation im Simulator abläuft. In der vorgestellten Lösung für GPSS/H (und für SLX analog) wird dies durch einen priorisierten Thread realisiert. Die zu wählende Priorisierung hängt von verschiedenen Interna des Simulators ab; insbesondere davon, wann Ereignisse in die zukünftige Ereigniskette eingetragen werden und wie gleichzeitige Ereignisse behandelt werden.

Das Arbeitsprinzip dieser Planungsroutinen besteht darin, das jeweils nächste Ereignis aus der zukünftigen Ereignisliste des lokalen Federates zu bestimmen (bei der zeitschrittgesteuerten Variante den nächsten Zeitschritt), den entsprechenden Zeitfortschritt bei der RTI zu beantragen und bis zur Antwort den lokalen Zeitfortschritt zu blockieren. Die Gewährung des Antrages (*TimeAdvanceGrant*) obliegt der RTI, welche die Abhängigkeiten der Federates und die verwendeten Synchronisationsprotokolle (optimistisch, konservativ mit individuellen dynamischen Lookaheads, etc.) berücksichtigt.

Die RTI teilt daraufhin mit, ob ein zeitlich früher gelegenes Ereignis (z.B. eine Änderung eines „fremden“ Objektattributs) zu berücksichtigen ist oder der beantragte Zeitraum gewährt werden kann. Nun kann die Planungsroutine zum gewährten Zeitpunkt fortschreiten und ggf. relevante Attributsänderungen durchführen. Bis zum nächsten Fortschritt der lokalen Zeit übernimmt das eigentliche Simulationsmodell die Kontrolle.

5.2 Datendarstellung und -austausch

Die Realisierung der Darstellung von externen Daten im lokalen Federate bereitete keinem der untersuchten Simulatoren größere Probleme. Es sei hier allerdings anzumerken, daß bisher nur statische Objekte untersucht wurden (logische Schalter, Variablen, etc.). Eine Lösung für die Darstellung von Informationen über dynamische Objekte (z.B. der Transaktionen in GPSS/H) ist Gegenstand laufender Untersuchungen.

5.3 Programmierungstechnische Umsetzung der Anforderungen

Für GPSS/H (Version für MS-DOS) ist es gelungen, sämtliche der oben erwähnten konzeptionellen Forderungen zu erfüllen. Eine direkte Anbindung an die RTI über die für die Intel-Plattform vorliegende Dynamic Link Library (DLL) war jedoch durch die beschränkten Kommunikationsmöglichkeiten von GPSS/H nicht realisierbar, da die GPSS/H-Funktionen zum Einbinden externen C-Codes keine Nutzung bereits compilierter DLL's zulassen. Eine GPSS/H-kompatible javabasierte Implementation mit HLA-Schnittstelle, die die o.g. Einschränkungen aufhebt, ist in Vorbereitung [5].

In der GPSS/H Implementation existieren 3 getrennte Module, die als an HLA angelehnte Federates agieren, sich jedoch nicht an die Interface Spezifikation halten und das RTI nicht nutzen. Sie synchronisieren ihre lokalen Zeiten unter Nachbildung der RTI-Funktionalität und eines konservativen Synchronisationsprotokolls unter Zuhilfenahme der oben erwähnten Planungsroutine. Aufgrund der eingeschränkten Kommunikationsmöglichkeiten von GPSS/H erfolgt der Datenaustausch über Dateien, womit die zweite konzeptionelle Forderung nur unter Umgehung der HLA-Interface-Spezifikation erfüllt werden konnte. Dies stellt nach unseren Erkenntnissen auch das grundsätzliche Problem bei vielen klassischen Simulatoren dar.

Eine Möglichkeit, Simulatoren trotz der oben aufgeführten prinzipiellen Beschränkungen an die RTI anzubinden, besteht in der Verwendung von Gateway-Programmen. Die Kommunikation zwischen dem Gateway-Programm, welches in C++ geschrieben werden könnte, und dem Simulator würde über Dateien bzw. Pipes erfolgen. Potentielle Funktionsaufrufe des Simulators würden in entsprechende Einträge in Dateien umgesetzt, vom Gatewayprogramm entsprechend ausgewertet und an die RTI weitergeleitet. Ebenso würden Funktionsaufrufe der RTI an den Simulator bei dem Gatewayprogramm erfolgen, welches dem Simulator die Information über Dateieinträge übermittelt. Dieses Konzept hat jedoch mehr theoretischen Charakter, da die Effizienz aufgrund des Pollings der Dateien bzw. Pipes nach neuen Daten sehr unbefriedigend wäre.

Erst bei der Untersuchung von SLX, eines relativ neuen Simulationstools für die Windows 95 / NT Betriebssysteme, gelang es, sämtliche Forderungen zu erfüllen und SLX als vollwertiges Federate zu verwenden. Es kamen das gleiche Modell und die gleichen Lösungsansätze für die konzeptionellen Anforderungen wie in der GPSS/H Implementation zur Anwendung, wobei wir in diesem Fall die Fähigkeit von SLX zur Einbindung externer DLLs nutzen konnten. Da die Aufrufkonventionen jedoch von SLX limitiert werden, wurde eine Wrapper-DLL entwickelt, die zwischen Aufrufen von SLX und eigentlichen RTI-Aufrufen vermittelt und gleichzeitig die Funktion des Federate Ambassadors übernimmt. Diese DLL löst auch gleichzeitig das Problem, daß man durch das C++ API an diese Sprache gebunden ist, und kann gleichzeitig als Basis für die RTI-Ankopplung von anderen Simulatoren dienen, die DLLs einbinden können.

6 Anforderungen für weitergehende HLA-Techniken

Beabsichtigt man die Nutzung von fortgeschrittenen Techniken der HLA (z.B. das Speichern und Laden des Zustands der Federation Execution), ergeben sich weitere, hier noch nicht diskutierte Probleme. Die entsprechende Funktionalität wird erst in späteren Versio-

nen der RTI implementiert, weshalb sich weitergehende praktische Untersuchungen verzögern. Diese zusätzlichen Anforderungen werden stärker als die oben diskutierten von der Unterstützung im konkreten Simulator abhängen. So ist z.B. in GPSS/H das Abspeichern und Wiederherstellen des Modellzustandes prinzipiell möglich (*Checkpoint*-Befehl.

7 Zusammenfassung und Ausblick

Es wurde gezeigt, daß es bei Erfüllung der hier vorgestellten Anforderungen möglich ist, konventionelle stand-alone Simulatoren als HLA-konforme Federates zu nutzen, wobei die Effizienz der vorgestellten Methoden beträchtlich differiert. Auch sind gewisse „Kosten“ wie z.B. für Koordination und Kommunikation, die für die Vorzüge wie Modularität und Wiederverwendbarkeit anfallen, zu berücksichtigen. Wurde für ein Simulationstool der Initialaufwand zur Erstellung einer RTI-Schnittstelle erbracht, so reduziert sich der eigentliche, HLA-bedingte Aufwand auf die Implementation der Modellkomponenten in den einzelnen Federates und die Aufstellung der Objektmodelle. An der Erstellung einer solchen allgemeinen Lösung für den Simulator SLX wird derzeit gearbeitet. Generell ist zu erwarten, daß die HLA aufgrund ihrer Flexibilität neue Anwendungsfelder, -techniken und -qualitäten erschließen kann; für klassische Simulationstools erlaubt dies einen vielversprechenden Schritt in Richtung wiederverwendbarer und interoperabler Simulationsmodelle.

Literatur

[1] Defense Modeling and Simulation Office (DMSO). *High Level Architecture Homepage.* URL http://www.dmso.mil/projects/hla

[2] Mehl, H. *Methoden verteilter Simulation.* Vieweg Verlag, Braunschweig, 1994

[3] Fujimoto, R. *Parallel Discrete Event Simulation.* In Communications of the ACM, 1990, no. 10, pp. 30-53

[4] Klein, U. *Zivile Anwendungspotentiale der High Level Architecture (HLA).* Symposium Neue Technologien in der wehrtechnischen Simulation, 30.09.-02.10.1997, Mannheim, in Vorbereitung

[5] Klein, U., S. Strassburger, J. Beikirch. *Distributed Simulation with JavaGPSS based on the High Level Architecture*. International Conference on Web-based Modeling and Simulation, Jan. 11.-14. 1998, San Diego, in Vorbereitung