HLA basierte verteilte Simulationsmodelle für die Fertigung

Thomas Schulze[1]
Ulrich Klein, Steffen Strassburger, Klaus Christoph Ritter[2]
Eberhard Blümel, Marco Schumann[3]

Zusammenfassung

Die High Level Architecture (HLA) ist ein Ansatz der verteilten Simulation. Im Mittelpunkt des Beitrags steht die Nutzung dieser Architektur für verteilte Simulationsmodelle von Fertigungsprozessen. Es wird eine prototypische *Federation* aus heterogenen *Federates* vorgestellt, wobei sich die beteiligten Federates hinsichtlich ihrer Funktionalität, ihrer Implementationssprache und ihres lokalen Zeitmanagements unterscheiden. Im Beitrag werden die Ziele und die Struktur der Federation sowie die Funktionalität der Federates erläutert. An dieser Federation werden besonders die Anforderungen an die Interoperabilität von verteilten Simualtionsmodellen aufgezeigt.

1 Verteilte Simulation in der Fertigung

Die Simulationstechnik ist in der Produktion und Logistik in den letzten Jahren zu einem unverzichtbaren Werkzeug bei der Gestaltung komplexer Unternehmensprozesse geworden. Die gestiegene Leistungsfähigkeit der Hard- und Softwaresysteme ermöglichen das Betreiben „virtueller Fabriken" in einem sehr hohen Detailliertheitsgrad. Die Arbeit am Simulationsmodell unterstützt die Lösungsfindung für das existierende oder das erst zu planende reale System. Die Einsatzgebiete der Simulation erstrecken sich dabei von der Produktentwicklung über die Fabrikplanung, die Strukturierung der Fertigung, Personaleinsatzplanung, Dimensionierung und Steuerung der Material- und Informationsflüsse bis zur Distribution der Fertigprodukte.

Für diese Applikationsgebiete wurden in den vergangenen Jahren leistungsstarke Simulationswerkzeuge entwickelt. Beispielhaft seien hier die Systeme SIMPLE++, CREATE!, PERSIMO, ARENA, FACTORY/AIM und QUEST aufgeführt. Zu diesen Werkzeugen sind umfangreiche Bibliotheken, die auf das jeweilige Anwendungsgebiet

[1] Technische Universität Dresden, Institut für Informationssysteme; e-mail: tom@isg.cs.uni-magdeburg.de
[2] Otto-von-Guericke Universität Magdeburg, Institut für Simulation und Graphik; e-mail: {uklein@isg, strassbu@sunpool, kcritter@sunpool}.cs.uni-magdeburg.de
[3] Fraunhofer Institut für Fabrikbetrieb und -automatisierung (IFF), Magdeburg; email: bluemel@iff.fhg.de

ausgerichtete Modellelemente oder Funktionen zur Ergebnisgewinnung und -präsentation beinhalten, entstanden. Dennoch werden der hohe Aufwand für die Akquisition von Eingabedaten, die Modellerstellung und die adäquate Ergebnisdatenpräsentation als Hauptkritikpunkte des Simulationseinsatzes in der Fertigung gesehen. Insbesondere vor dem Hintergrund der wachsenden Anforderungen des Marktes, der von den Unternehmen immer schnelleres und zielsicheres Handeln verlangt, stellt sich die Frage nach den Konsequenzen für neue Einsatzkonzepte der Simulation.

Immer kürzere Produktlebenszyklen, die Vielzahl kundenspezifischer Produkte, neue Technologien zwingen die Unternehmen zur permanenten Neu- oder Umgestaltung ihrer Produkte, Fertigung und Prozesse. Es entsteht die **Forderung nach Wiederverwendbarkeit** von Simulationsmodellen bzw. nach Methoden zu deren zeit- und kostensparenden Erstellung bzw. Anpassung. Insbesondere ist die Datengewinnung und Modellierung komplexer Fabrikanlagen sehr zeit- und änderungsintensiv. Einsparungsmöglichkeiten ergeben sich durch die Einbindung vorhandener Anlagenmodelle des Anlagenlieferanten.

Durch die Globalisierung der Produktion werden selbst kleine und mittlere Unternehmen in den weltweiten Wertschöpfungsprozeß von Produktionsnetzwerken einbezogen, sei es als Lieferant, Konstrukteur oder Servicedienstleister. Simulationsmodelle zur Abbildung derartiger Produktionsnetzwerke müssen in der Lage sein, mit existierenden oder zu erstellenden **Modellen der beteiligten Unternehmen zu kommunizieren**. Damit verbunden ist möglicherweise die Unterstützung unterschiedlicher Modellarten (kontinuierliche oder diskrete Modelle). Ein weiterer wichtiger Punkt ist die flexible Anpassung des Modells hinsichtlich neu hinzukommender oder ausscheidender Partner. Damit verbunden ist mitunter auch die Untersuchung sich ändernder Simulationsziele.

Bisherige Paradigmen der Simulationstechnik führen bei der Lösung der beiden angeführten Problembereiche zu sehr komplexen monolithischen Modellen, die auch als Einzweckmodelle bezeichnert werden können, welche nur mit sehr großem Aufwand auf neue Aufgabenstellungen anzupassen sind. Der Weg zu neuen Lösungsmöglichkeiten führt über die Schlagworte **verteilte Simulation**, Wiederverwendbarkeit von Simulationsmodellen sowie Interoperabilität zwischen Simulationsmodellen. Unter **verteilter Simulation** wird eine zeitlich parallele Nutzung von gegebenenfalls heterogenen Simulationsmodellen auf unterschiedlichen Rechnern verstanden. Komplexe Modelle werden in eigenständige Simulationsmodelle aufgeteilt, wobei jedes selbständige Modell seinen eigenen Speicherbereich, seinen eigenen Prozeß und sein eigenes Zeitmanagement (z.B. eigene Ereignislisten) besitzt. Das Hauptziel bei dieser Form der verteilten Simulation liegt in der parallelen Nutzung von verteilten Simulationsmodellen, die nicht zu einem monolithischen Modell zusammengefaßt werden können.

Mit der High Level Architechture (HLA) steht seit 1997 eine Architektur zur Umsetzung der verteilten Simulation allen Interessenten offen. Entsprechend dem militärischen Ursprung dominieren in den bisherigen HLA-Applikationen militärische Anwendungsgebiete. Mit den folgenden Abschnitten wird gezeigt, daß dieses Konzept auf Modelle aus dem Bereich Fertigungsprozesse übertragbar ist und somit ein weiteres ziviles Anwendungsfeld erschlossen wird.

2 Verteilte Simulation auf der Basis von HLA

Die Motivation der High Level Architecture ist hauptsächlich in der Unterstützung der Wiederverwendbarkeit und Interoperabilität von Simulationsmodellen zu sehen, wobei die Schwierigkeiten mit monolithischen Modellen bei Änderungen der Funktionalität oder Konnektivität umgangen werden können [1].

Die HLA, welche selbst wiederum ein Teil eines umfassenderen technischen Rahmenwerks ist (neben den *Conceptual Models of Mission Space* und den *Data Standards*), besteht aus den drei Hauptkomponenten:

- HLA Schnittstelle (*HLA Interface*), 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.
- HLA Regeln (*HLA Rules*), welche das Verhalten von Federates und der *Federation* vorschreibt; letztere kann als ein Vertrag zur Durchführung eines Simulationslaufes (*Federation Execution*) zwischen den Federates angesehen werden.
- HLA Objektmodellschablone (*HLA Object Model Template, OMT*), welche die Art und Weise der Definition und Dokumentation der Objektmodelle der Federates und der Federation festlegt.

Die HLA nutzt eine objektorientierte Sichtweise, um die in einer Federation abgebildeten (simulierten) Dinge zu beschreiben. Abweichend von der klassischen OO-Systematik besitzen Objekte zwar Attribute als statische Elemente, dynamische Elemente (üblicherweise Methoden genannt) werden allerdings als eigenständige sogenannte *Interactions* dargestellt, welche ebenfalls über eine Hierarchie und Eigenschaften (Parameter genannt) verfügen und nicht-persistente, zu einem Zeitpunkt auftretende Ereignisse darstellen.

Jedes Federate hat ein *Simulation Object Model* (*SOM*) aufzuweisen, das gemäß dem OMT zu dokumentieren ist. Im SOM wird beschrieben, welche Objekt- und Interaktionsklassen das Federate zu modellieren / publizieren in der Lage ist und welche es abonnieren (in Zukunft übermittelt bekommen) möchte. Das SOM ist auch Hauptentscheidungsgrundlage bei einer Anfrage an ein Object Model Repository. ob ein Federate ggf. für eine zu bildende Federation (wieder)verwendet werden kann. Die Federation weist ein eigenes Objektmodell auf (*Federation Object Model, FOM*), in welchem zusammengefaßt wird, welche Objekt- und Interaktionsklassen im gemeinsamen Ereignisraum auftreten können. Diese Information wird auch von der Runtime Infrastructure ausgewertet.

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, [4]).

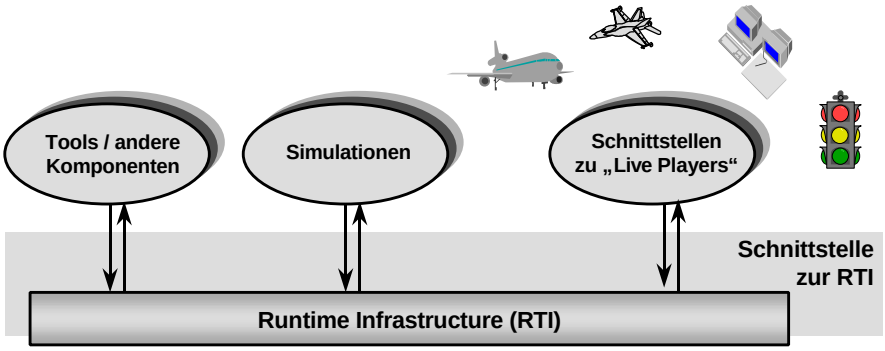

Abbildung 1: Funktionale Übersicht über die High Level Architecture

Das HLA Zeitmanagement erlaubt es, die Art und Weise des lokalen Zeitfortschrittes gegenüber anderen Federates transparent zu halten. Zwischen zeitschrittgesteuerten, ereignisorientierten, kontinuierlichen und echtzeitgetriebenen Federates kann eine Interoperabilität hergestellt werden, die auch konservative und optimistische Synchronisationsverfahren nebeneinander erlaubt.

Primär wird das zeitliche Verhalten eines Federates dabei durch zwei Eigenschaften charakterisiert:

- *time constrained*: dieser Schalter gibt an, ob das Federate durch den Zeitfortschritt der anderen Federates beeinflußt wird.
- *Time regulating*: hiermit kann eingestellt werden, daß die RTI die Zeit des Federates bei der Berechnung des Zeitfortschritts anderer Federates berücksichtigt.

Beispiele für (*regulating, constrained*) sind konservativ (ALSP) bzw. aggressiv (Time Warp) synchronisierte Simulationen. Passive Beobachter oder Federation Management Tools werden durch (*not regulating, constrained*) beschrieben, während Echtzeitsimulationen entweder durch (*not regulating, not constrained*) oder im Zusammenspiel mit konservativen Federates (*regulating, not constrained*) charakterisiert werden.

Der Datenaustausch zwischen den Federates wird durch ein sogenanntes Interessenmanagement etabliert, durch das die einzelnen Federates bei Eintritt in die Federation Execution mitteilen, welche Informationen (Objekt- und Interaktionsklassen) sie publizieren (modellieren und bekanntgeben) können bzw. welche Informationen sie abonnieren wollen. Dadurch ist es der RTI zur Laufzeit möglich festzustellen, ob und zu welchen Federates Updates zu liefern sind.

Das Data Distribution Management (DDM) erlaubt eine noch genauere dynamische Eingrenzung der zu übertragenden Daten über den Wertebereich der Objektattribute (z.B. Einschränkung auf alle sichtbaren Objekte eines abonnierten Typs). Diese gegenüber den Vorgängertechnologien neue Eigenschaft kann zu einer erheblich geringeren Netzlast führen.

Die RTI-Software setzt intern auf TCP/IP auf und ermöglicht damit eine umfassende Flexibilität von der Verteilung der Federates bis hin zur Wahl des Kommunikationsmediums.

3 Implementation einer HLA Federation *Fertigung*

3.1 Ziele der Federation

Die Implementierung der HLA-Federation *Fertigung* erfolgte unter dem Gesichtspunkt einer Demonstration, wobei gewonnene Erkenntnisse aus dieser Implementation in reale Projekte einfließen werden [3]. Die Ziele dieser Federation werden in zwei Klassen eingeteilt: allgemeine, welche sich auf die Anwendung von HLA in der verteilten Simulation beziehen und spezielle, welche die Ergebnisse für die verteilte Simulation in der Fertigung betreffen.

3.1.1 Allgemeine Zielstellungen

Nutzung heterogener Federates: Alle drei Federates sind in unterschiedlichen Sprachen implementiert. Hierbei kann die Interoperabilität HLA-basierter Software über Programmiersprachen sowie Hardware- und Betriebssystemplattformen hinweg demonstriert werden.

Zeitliche Synchronisation während der Simulation: Das Federate *Simulator* basiert auf einer ereignisorientierten Zeitfortschreibung. Im Gegensatz dazu operiert das Federate *Disponent* auf einer realzeitproportionalen Zeifortschreibung, während das Federate *Visualisierung* über ein unabhängig einstellbares zeitliches Verhalten verfügt (je nach Zielstellung Echtzeit oder Fertigungszeitfortschritt). Eine zeitliche Synchronisation wird unter Verwendung des *Time Managements* des RTI vorgenommen. Hierbei kann die Interoperabilität und Transparenz unterschiedlicher Zeitfortschrittsverfahren auf Basis von HLA untersucht werden.

Verteilung der Federates auf unterschiedliche Rechner: Jedes Federate läuft auf einem eigenem Rechner. Das Federate *Simulator* läuft unter Windows95 auf einem PC, das Federate *Visualisierung* auf einem Java-fähigen Rechner (alle Plattformen) und das Federate *Leitstand* operiert auf einem Rechner mit C++-Compiler (PC / Workstation).

Kommunikation über das Internet: Die Kommunikation der beteiligten Federates erfolgt über Internet-Protokolle. Damit soll der Nachweis von flexiblen Anwendungen in unterschiedlichen Netzwerken (sowohl LAN- als auch WAN-, Fest- als auch Wählverbindungen) erfolgen. Der Standort der Federates ist somit von untergeordneter Bedeutung.

Dynamische Strukturveränderungen der Federation: Unter dynamischen Strukturveränderungen der Federation wird das An- bzw. Abmelden von Federates während der Federation Execution verstanden. Das Verhalten der Federates bzw. der Federation z.B. bzgl. Zeitsynchronisation und Datenaustausch wird untersucht.

3.1.2 Spezielle Zielstellungen

Fertigungskenngrößen: Mit dieser Federation sollen statistische Parameter für ausgewählte Kenngrößen für Fertigungsprozesse ermittelt werden. Hierzu gehören z.B. die Anzahl der gefertigten Teile je Schicht, die Verweilzeit der Teile im Fertigungsbereich und die Entwicklung des Bestandes (WIP). Diese Zielstellungen werden von dem Federate *Simulator* unterstützt.

Prozeßvisualisierung: Die Animation des simulierten Fertigungsprozesses ist eine weitere lokale Zielstellung. Das Federate *Visualisierung* ist für diese Darstellungsform verantwortlich.

Disponententraining: Ein Leitstand mit Disponent ist diesem Fertigungsbereich zugeordnet. Der Disponent trainiert unterschiedliche Strategien zur Störungsbeseitigung unter Echtzeitbedingungen.

3.2 Struktur der Federation

Die Federation besteht aus den drei Federates: *Simulator*, *Visualisierung* und *Leitstand*. Der *Simulator* ist in SLX [2,5] implementiert und simuliert den Fertigungsprozeß ereignisgesteuert. Das Federate *Visualisierung* gestattet eine online Betrachtung der simulierten Prozesse auf der Basis des Animationssystems Skopeo [4]. Der Zeitfortschritt in der *Visualisierung* paßt sich dem Federationzeitfortschritt an. *Leitstand* ist ein Federate, mit dem ein Nutzer in Echtzeit Eingriffe in den simulierten Prozeß vornimmt. Alle drei Federates werden in weiteren Abschnitten näher erläutert.

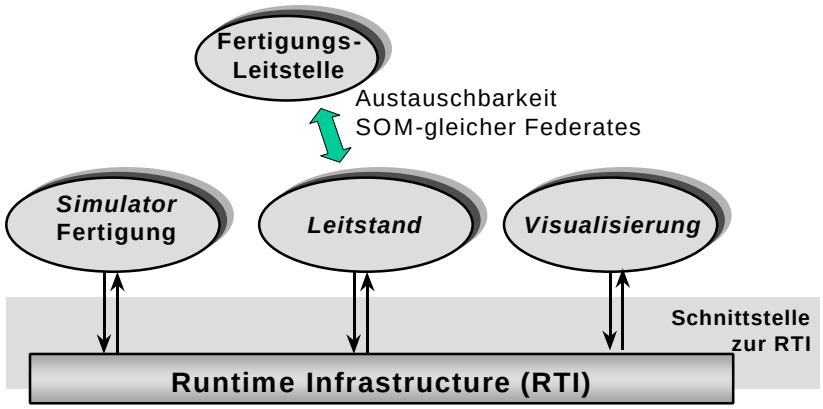

Abbildung 2: Funktionale Übersicht über die Fertigungsfederation

Eine interessante Eigenschaft ist in Abbildung 2 angedeutet: SOM-gleiche Federates können gegeneinander ausgetauscht werden, so z.B. eine Simulation gegen das Echtsystem (wie hier die Leitstandsimulation vs. Leitstand).

Als Runtime Infrastructure Software wird die Version 1.0 Release 3 des U.S. Defense Modeling & Simulation Office (DMSO) verwendet. Diese Software ist für zahlreiche

Unix-Varianten, Windows NT und Java verfügbar. Der RTI Executive Prozeß (RTIExec) ist als Dämon auf einer Maschine der Universität Magdeburg permanent verfügbar.

3.3 Objektmodell der Federation

Für die Federates als auch für die Federation wurden entsprechende Objektmodelle (SOMs bzw. FOM) vereinbart. In diesen Modellen sind Objekte und Interaktionen sowie deren Attribute und Parameter in einer Zahl von Tabellen definiert. Die Tabelle 1 gibt Auskunft über das Objektmodell der Federation, welche den gemeinsamen Ereignisraum beschreibt, in einer die OMT-Tabellen *Object Class Structure Table* und *Attribute/Parameter Table* zusammenfassenden Form.

Objekt	Attribute	Datentyp	Bedeutung
Maschine			Ressourcen in der Fertigung
	ID	Integer	Identifikationsnummer
	Status	{frei, belegt, blockiert, gestört}	Zustand einer Ressource
	Verarbeitete_Teile	Integer	Anzahl der verarbeiteten Teile
Transporter			Transporter in der Fertigung
	ID	Integer	Identifikationsnummer
	Kapazität	Integer	Transportkapazität
	Ladung	Integer	Belegte Einheiten
	Geschwindigkeit	Real	Geschwindigkeit
Quelle			Einschleusen von Teilen in den Fertigungsprozeß
	ID	Integer	Identifikationsnummer
	Erzeugt	Integer	Anzahl der eingeschleusten Teile
Senke			Ausschleusen von Teilen in den Fertigungsprozeß
	ID	Integer	Identifikationsnummer
	Vernichtet	Integer	Anzahl der ausgeschleusten Teile
Teil			Teile im Fertigungsprozeß
	ID	Integer	Identifikationsnummer
	Status	{bereit, inBearbeitung, fertig}	Zustand der Teile
	Geometrie	Form	Geometrieinformationen

Disponent	Lokation	Integer	Position des Teiles
			Disponent in der Fertigung
	ID	Integer	Identifikationsnummer
	Status	{aktiv, passiv}	Zustand des Disponenten

Tabelle 1: Funktionale Übersicht über die Fertigungsfederation

Für die Federation sind zwei Interaktionen vereinbart. Die Interaktion *Maschine_ausgefallen* beschreibt den Ausfall einer Maschine, und die Interaktion *Maschine_repariert* die Beendigung der Maschinenreparatur.

Interaktion	Sender	Empfänger	Parameter
Maschine_ausgefallen	Ressource	Disponent	ID_Maschine, Fehlerursache
Maschine_repariert	Disponent	Ressource	ID_Maschine

Tabelle 2: Funktionale Übersicht über die Fertigungsfederation

3.4 Zeitliche Synchronisation in der Federation

Aus der möglichen (Teil-)Zusammensetzung der Föderation ergeben sich unterschiedliche Anforderungen an die zeitliche Synchronisation zwischen den beteiligten Federates. Kombination und Anforderungen an die zeitliche Synchronisation zeigt Tabelle 3.

Federate			Bemerkung
Simulator	Visualisierung	Leitstand	
X	X	X	*Leitstand* bestimmt den zeitlichen Fortschritt (Standardkonfiguration)
X		X	*Leitstand* bestimmt den zeitlichen Fortschritt
X	X		*Simulator* und *Visualisierung* arbeiten nach „so schnell als möglich“
X			*Simulator* bestimmt den zeitlichen Fortschritt
	(X)	(X)	*Lediglich zu Testzwecken; Animation zeigt Aktionen des Leitstandes in Echtzeit*

Tabelle 3: Funktionale Übersicht über die Fertigungsfederation

Auf die Gestaltung des lokalen Zeitmanagements der einzelnen Federates und der jeweiligen Handhabung durch die Runtime Infrastructure wird in den nachfolgenden drei Kapiteln eingegangen, die die einzelnen Federates näher beschreiben.

Interessant hierbei ist, daß nicht alle Federates zum Funktionieren der Federation nötig sind; vielmehr ergibt sich allein aus der Anwesenheit bestimmter Federates ein schlüssiges zeitliches Verhalten der anderen Federates der Federation.

Dieses Verhalten gilt auch im Falle dynamischen Bei- und Austritts, so z.B., wenn während eines zeitaufwendigen as-fast-as-possible-Simulationslaufes mittels des Visualisierungstools der Zwischenstand dargestellt werden (und anschließend ausgetreten werden) kann.

4 Das Federate *Simulator*

Das Federate *Simulator* simuliert einen einfachen Fertigungsprozeß. Bei dem simulierten Fertigungsbereich handelt es sich im wesentlichen um eine Fließbandfertigung. Eine Quelle schleust die zu bearbeitenden Teile in das System ein. Die Teile müssen drei Arbeitsgänge durchlaufen, bevor sie fertiggestellt werden. Vor jeder Maschine befindet sich eine Staustrecke, die maximal fünf Teile aufnehmen kann. Ist eine Staustrecke vollständig mit wartenden Teilen besetzt und wird auf einer davor befindlichen Maschine ein weiteres Teil fertiggestellt, so verbleibt das fertiggestellte Teil in der Maschine, die damit für die Bearbeitung weiterer Teile blockiert ist. Der zweite Arbeitsgang ist besonders zeitintensiv. Daher existieren zwei Maschinen, die diesen Arbeitsgang ausführen können. Die Teile werden auf diejenige Maschine verteilt, auf deren Staustrecke am meisten freier Raum vorhanden ist. Die Struktur der Fertigung ist in Abbildung 3 dargestellt.

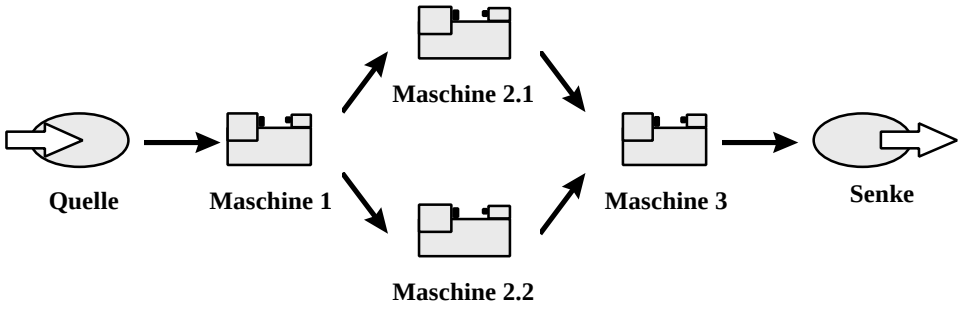

Abbildung 3: Der simulierte Fertigungsbereich

Zur Modellierung des Fertigungsbereich werden aktive und passive Objekte verwendet. Aktive Objekte besitzten eine eigene Ablaufsteuerung, passive Objekte werden hingegen nicht. Die zu bearbeitenden Teile wurden als passive Objekte modelliert, die durch die als aktiven Objekte modellierten Maschinen bewegt werden. Tabelle 4 zeigt die verwendete Objekthierachie.

Objekttyp	1. Hierachieebene	2. Hierachieebene
Aktives Objekt	Ressource	Maschine
		Lager

		Quelle
		Senke
Passives Objekt	Bewegliches Teil	

Tabelle 4: Die verwendete Objekthierachie

Maschinen können durch zufällig auftretende Störungen ausfallen. In diesem Fall wird die Interaktion *Maschine_ausgefallen* ausgelöst. Falls das Federate *Leitstand* an der Federation teilnimmt, erfolgt die Behebung der Störungen nicht im Federate *Simulator*. Die entsprechende Maschine bleibt solange gestört, bis vom *Leitstand* eine Interaktion *Maschine_repariert* ausgelöst wurde. Damit die Simulation bei einem Maschinenausfall nicht verklemmt falls das Federate *Leitstand* nicht an der Federation teilnimmt, wird in diesem Fall nicht auf eine Interaktion *Maschine_repariert* gewartet, sondern nach einer festzulegenden Zeit wird die Maschine als repariert betrachtet.

Kommunikation

Die Fertigungssimulation publiziert die Objekte und ihre Zustandsänderungen im Modell. Diese Zustandsänderungen werden vom Federate *Visualisierung* dargestellt. Geht eine Maschine in den Zustand *gestört* über, so wird die Interaction *Maschine_ausgefallen* gesendet und im *Leitstand* eine entsprechende Meldung erzeugt.

Zeitfortschritt

Das zeitliche Verhalten der Fertigungssimulation kann durch die Begriffe *time constrained* und *time regulating* beschrieben werden: der lokale Zeitfortschritt ist vom Zeitfortschritt anderer Federates (Leitstand) abhängig, wirkt aber auch regulierend auf den anderer Federates (der Visualisierung). Fehlt es an einem beschränkenden Federate, so wird die Simulationsgeschwindigkeit nur durch die verwendete Hardware bestimmt.

5 Federate *Visualisierung*

Als Visualisierungskomponente dieses HLA-Federation wird als Grundlage Skopeo verwendet. Dieses Java-basierte Tool ist für den Einsatz im Web konzipiert und läuft als Applikation bzw. Applet. Skopeo stellt standardmäßig eine 2D-Visualisierungskomponente zur Verfügung. Soll eine Visualisierung im 3D-Bereich erfolgen, muß auf externe 3D-Viewer zurückgegriffen werden. Skopeo unterstützt hierbei grundsätzlich alle VRML-Browser, die ein External Authoring Interface (EAI) zur Verfügung stellen. Wenn diese 3D-Viewer benutzt werden sollen, muß Skopeo als Applet in eine Web-Seite, in der auch der VRML-Browser integriert ist, eingebunden werden. Das Applet und der VRML-Browser (in diesem Fall der CosmoPlayer von SGI) können dann Animationsdaten über eine JavaScript-Schnittstelle im Inneren des Web-Browsers austauschen.

Die Verwendung der Web-gestützten Visualisierung hat neben den generellen Vorteilen im Bereich von Installation, Wartungsfreundlichkeit und Verfügbarkeit Nachteile, die insbesondere den Einsatz im Bereich verteilter Systeme erschweren. Dazu gehört insbesondere die Nutzung von Ressourcen außerhalb der Client-Rechner. Um das „Sandbox-Environment“ des Web-Browsers, das den Client vor Attacken von außen

schützt, verlassen zu können, muß ein recht aufwendiger Mechanismus installiert werden. Da das Applet nur mit seinem Hostserver kommunizieren darf, ist ein Kommunikationsrelais auf diesem Server unumgänglich. Für diesen Zweck wurde eine Kommunikationsstruktur auf CORBA-Basis installiert.

Im hier dargestellten Fall kommuniziert das Skopeo-Applet mit dem Kommunikationsrelais auf dem Hostserver. Gleichzeitig kommuniziert der Skopeo-RTISchnittstelle mit dem selben Kommunikationsrelais. Der Datenaustausch zwischen Skopeo und dem RTI wird somit über zwei Zwischenstufen realisiert.

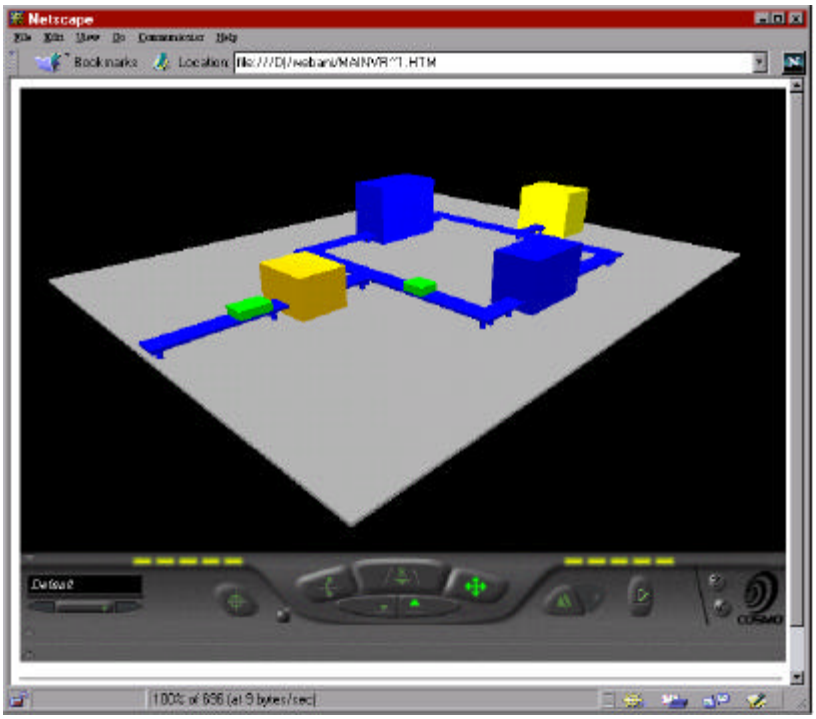

Abbildung4: Der simulierte Fertigungsbereich

Kommunikation:

Dieses Federate abonniert sämtliche Objektklassen (bzw. deren Attribute), die sich auf Animationsereignisse beziehen. Ebenso werden Interactions abonniert, um diese Ereignisse ebenfalls darstellen zu können. Da der Viewer in dieser Federation passiv agiert, publiziert er keine Attributsänderungen oder Objekte.

Zeitfortschritt:

Hier ist zwischen der RTI-Seite und der Visualisierungskomponente von Skopeo zu unterscheiden: während der Zeitfortschritt gegenüber der Federation durch den Zeitfortschritt in der Fertigungssimulation bestimmt wird (daher *time constrained* ist), jedoch aufgrund der rein passiven Beobachterkontrolle nicht zeitbeeinflussend für andere

Federates ist (daher *not time regulating*), wird die Visualisierungsseite vom Nutzer bestimmt. Hier kann zwischen einer Visualisierung des aktuellen Systemzustandes (Kopplung der Visualisierung mit der Federation) und einer Wiedergabe aus dem sich während der Federation Execution ständig aufbauenden Animationspuffer ähnlich einem Videorekorder gewählt werden [4]. Für eine aktive Rolle, z.B. wenn über interaktive VRML2-Elemente in der Visualisierung Aktionen für die Fertigungssimulation ausgelöst werden sollen, wäre eine kausalitätserhaltende direkte Kopplung der Federation- und Visualisierungszeit (*time regulating*) erforderlich.

6 Federate *Leitstand*

Das Federate *Leitstand* modelliert einen Fertigungsleitstand mit einem Disponenten. Mit diesem Federate werden interaktive Eingriffe eines Menschen in die Federation realisiert (man-in-the-loop-Simulation). Der Disponent an einem Leitstand hat mehrere Prozesse zu beaufsichtigen. Bei einem auftretenden Fehler an einer Maschine aus der Federation *Simulator* generiert das Federate ein Signal für den Disponenten. In einem Bildschirmfenster wird ihm die ausgefallene Maschine und der Art des Ausfalls mitgeteilt. Aufgrund dieser Informationen hat der Disponent einen Reparaturauftrag auszulösen. Nach der Erledigung dieses Auftrages ist die Maschine zur Weiterarbeit zu veranlassen.

Kommunikation

Das Federate *Leitstand* abonniert Objektklassen aus dem Bereich der Fertigung. Es empfängt die Interaktion *Maschine_ausgefallen* und sendet nach Erledigung eines Reparaturauftrages die Interaktion *Maschine_repariert*.

Zeitfortschritt

Die Zeit in der Federation soll bei Anwesenheit des Federates *Leitstand* echtzeitproportional fortschreiten. Der Disponent als „Man-in-the-loop“ muß unter Echtzeitbedingungen reagieren. Für spezielle Trainingszwecke ist das Verhältnis zwischen Echtzeit und modellierter Zeit einstellbar, womit für den Disponenten eine Streßumgebung eingestellt werden kann.

Für die HLA-Zeitcharakterisierung bedeutet das die Eigenschaft *time regulating*, da der Leitstand Einfluß auf den Zeitfortschritt der Fertigungssimulation hat und somit eine Echtzeittaktung in die Federation einbringt. Als Echtzeitkomponente ist weiterhin von *not time constrained* auszugehen, da die Synchronisation ausschließlich mit der Echtzeituhr erfolgt. Möchte man erreichen, daß im Falle von Performanzproblemen der Fertigungssimulation der Leitstand wartet und damit die Kausalität gewahrt bleibt, ist auch *time constrained* möglich.

7 Ausblick

Die mit dieser Federation gesammelten Erfahrungen zeigen, daß HLA ein möglicher Weg ist, um die Wiederverwendbarkeit und Verteilung von Simulationsmodellen auf eine neue Stufe zu setzen. Durch diesen Beitrag sollen andere Simulationisten ermuntert werden, HLA-basierte verteilte Simulationsmodelle für zivile Anwendungsgebiete zu entwickeln.

Die an dem Beitrag beteiligten Institutionen werden, unter Verwendung der bisherigen Erkenntnisse, HLA in weiteren zivilen Anwendungsfeldern nutzen.

Es bleibt zu hoffen, daß immer mehr Entwickler von Simulations- und Animationssoftware den Initialaufwand der Integration einer HLA-Schnittstelle erbringen, um die vollen Vorteile interoperabler verteilter Simulation nutzen zu können.

8 Literatur

[1] Department of Defense (US). *High Level Architecture Interface Specification, Version 1.2, 13.8.1997*. Verfügbar online unter http://hla.dmso.mil.

[2] Henriksen, J.O., An Introduction to SLX. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopolous, pp. 502-507.

[3] Klein, U., S. Straßburger. *Die High Level Architecture: Anforderungen an interoperable und wiederverwendbare Simulationen am Beispiel von Verkehrs- und Infrastruktursimulationen*. 11. Symposium Simulationstechnik ASIM 97, 11.11.-14.11.1997, Dortmund.

[4] Ritter, K.C., U. Klein, S. Straßburger, M. Diessner. Web-basierte Animation verteilter Simulationen auf der Basis der High Level Architecture (HLA). In *Proceedings der Tagung Simulation und Visualisierung 1998*, 5.-6. März 1998, Magdeburg.

[5] Straßburger, S. *Integration klassischer Simulationstools in die High Level Architecture*. Diplomarbeit, Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg 1998, Magdeburg.